SIMPLE ASSEMBLY LINE BALANCING PROBLEM: A PETRI NET APPROACH

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of
Dokuz Eylül University
In Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy in Industrial Engineering, Industrial Engineering Program

by

Özcan KILINÇCI

March, 2003
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We certify that we have read this thesis **"Simple Assembly Line Balancing Problem: A Petri Net Approach"** completed by **Özcan KILINÇCI** under supervision of **Prof. Dr. G. Miraç BAYHAN** and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. G. Miraç BAYHAN

Doç. Dr. Semra Tunalı

Y. Doç. Dr. Adil Alpkoçak

Yrd. Doç. Dr. Aşkıner GÜNGÖR

Yrd. Doç. Dr. Mehmet ÇAKMAKÇI

Approved by the

Graduate School of Natural and Applied Sciences

Prof.Dr. Cahit HELVACI

Director

ABSTRACT

Much research has been devoted to the simple assembly line balancing problem-type 1 (SALBP-1). The objective is to minimize number of stations for a given cycle time. As they are known as NP-hard problems, approximate or heuristic approaches are extremely required for solving these problems. In this thesis, heuristic algorithms based on Petri nets (PNs) are proposed to solve SALBP-1. Although PNs have been widely used as mathematical and graphical tools in the area of assembly systems, a great deal of research attention has been rather focused on modeling, controlling, planning, task sequencing, and scheduling. The thesis involves particularly a review of the recent research works that applied to SALBP-1, and a review of PNs applications on manufacturing systems. In these works, classification frameworks are provided. Complexity analyses of the proposed algorithms are presented. A problem is considered to demonstrate the application of these algorithms. All iterations of solutions are given in details. The algorithms are coded in MATLAB, the language of technical computing, and are compared with each other and with existing algorithms on the sets of large test problems from the literature.

Keywords: Assembly line balancing, Petri nets, Manufacturing systems, Reachability analysis, *T*-invariants, *P*-invariants, SALBP-1.

ÖZET

Birinci tip basit montaj hattı dengeleme problemi (BMHDP-1) ile ilgili bir çok çalışma yapılmıştır. Amaç, verilen bir çevrim süresi için istasyon sayısını en küçüklemektir. NP-hard problemler olarak bilinen bu problemlerin çözümü için yaklaşık veya sezgisel yöntemler gerekmektedir. Bu tezde, BMHDP-1'i çözmek için Petri ağlarına dayalı sezgisel algoritmalar önerilmiştir. Petri ağları, matematiksel ve grafiksel bir araç olarak montaj sistemlerinde yaygın olarak kullanılmasına rağmen, araştırmaların büyük bir kısmında modelleme, kontrol, planlama, iş sıralama ve çizelgeleme konularında yoğunlaşma tercih edilmiştir. Tez, son yıllarda montaj hattı dengeleme problemleri konusunda yapılan çalışmaları ve Petri ağlarının üretim sistemlerindeki uygulamalarını kapsayan iki literatür taraması içerir. Bu çalışmalar belirli sınıflandırma yapıları içersinde düzenlenmişlerdir. Önerilen algoritmaların kompleksklik analizleri verilmiştir. Bu algoritmaların uygulanışını göstermek için bir problem tasarlanmıştır. Çözümlerin tüm aşamaları ayrıntılı olarak verilmiştir. Teknik bir hesaplama dili olan MATLAB programında kodlanan algoritmalar, birbirleriyle ve mevcut algoritmalar ile literatürdeki büyük test problem setleri üzerinde kıyaslanmıştır.

Anahtar Kelimeler: Montaj hattı dengeleme, Petri ağları, Üretim sistemleri, Ulaşılabilirlik analizi, *G*-değişmezleri (*T*-invariants), *K*-değişmezleri (*P*-invariants), BMHDP-1.

ACKNOWLEDGEMENTS

This thesis is dedicated to my father

Many people have helped me during the course of this research. I would like to thank my advisor, Prof. Dr. G. Miraç Bayhan, for her encouragement of this research and for her continuing support and guidance.

I gratefully acknowledge and thank the committee of this dissertation, Prof. Dr. Demir Aslan and Assist. Prof. Dr. Adil Alpkoçak, for their comments and suggestions.

I am truly grateful Prof. Dr. Armin Scholl for providing the benchmarking data and Prof. Dr. Zdenek Hanzelek for offering his MATLAB codes.

I would like to thank Prof. Dr. Hasan Eski, Head of the Department of Industrial Engineering, for his encouragement and support during my study.

Special thanks all professors in my undergraduate and graduate levels for their supports for my background.

Great thanks all my friends, especially Gökalp Yıldız, Özgür Eski, Ceyhun Araz, Özlem Uzun, and Şeyda Topaloğlu for being around me in my bad times and good times.

I sincerely appreciated my parents, Remzi and Türkan Kılınçcı, for all sacrifices they made to educate me. Without their love and support, I would not have been able to pursue my graduate studies. I would like to thank them for their ineffable support and prayers.

Özcan KILINÇCI

CONTENTS

Chapter Three

PETRI NETS AND APPLICATIONS IN MANUFACTURING SYSTEMS

Chapter Four

SIMPLE ASSEMBLY LINE BALANCING: A PETRI NET APPROACH

Chapter Five

APPLICATION

Chapter Six
CONCLUSION AND FUTURE RESEARCH

LIST OF TABLES

LIST OF FIGURES

Page

CHAPTER ONE
INTRODUCTION

1.1 Overview of the Problem

Today's industrial world includes many types of manufacturing systems. Lean manufacturing satisfies waste minimization. Just in time manufacturing aims zero inventories. Agile manufacturing provides ability of a manufacturing process to respond quickly to the demands of the customer. Computer-aided design and manufacturing systems use computers to integrate component design and processing instructions. Computer-integrated manufacturing integrates all aspects of manufacturing through computers. Flexible manufacturing systems combine flexibility with efficiency by minimizing the setup time, tool changing time, and labor cost. Assembly systems are part of all or some of these manufacturing systems.

Assembly systems are preferred in mass or batch productions. These systems include tasks, workstations, and sometimes a conveyor, AGV, or another transportation device for movement of products between the workstations, which are placed on parallel, serial, or U-shaped lines. A workstation, which has longer processing time than the others, determines the cycle time (or throughput rate) of the line. To design an efficient assembly system, tasks should be assigned to stations in a manner that the station's idle time would be zero. In the literature, this is called as assembly line balancing problem.

If, only one product is produced on the line, and times of tasks are known and constant, then the problem is defined as simple assembly line balancing problem (SALBP). SALBP is classified into three groups. If, the aim is *i)* to minimize the number of stations for a given cycle time, then the problem is type-1 (SALBP-1), *ii)* to minimize the cycle time for a fixed number of stations, then the problem is type-2

(SALBP-2), *iii)* to maximize the line efficiency for varying cycle time and number of stations, then the problem is type-E (SALBP-E). This thesis focuses on SALBP-1.

Assembly line balancing problem was first formulated as a mathematical program in 1950s. The problems with few tasks were optimally solved easily but when the number of the tasks on the line is bigger then the problem is complex. Optimally solving the problem is not possible because of the assembly line balancing problem falls into the NP-hard class of combinatorial optimization problems. Therefore, many heuristic techniques have been developed to solve SALBP-1. In the beginning, heuristics gave a sequence according the some priority rules or decision rules such as longest process time, shortest process time, and etc., were developed. Tasks were assigned using this sequence. But recently developed heuristics based on search techniques dynamically determine available tasks for each assignment operation. Branch and bound procedures, genetic algorithms, tabu search metaheuristics are preferred search techniques.

1.2 Research Objective

In this thesis, Petri nets (PNs) will be used to solve SALBP-1. PNs developed by Carl Adam Petri in 1960s, are mathematical and graphical tools to model, design, and analyze the discrete event systems. They have various application areas from the logical systems to industrial systems and from the software engineering to management operations. Many industrial problems such as designing, scheduling, deadlock detection and prevention, and controlling have been solved using PNs. To the best of our knowledge, this is the first study that uses PNs to solve SALBP-1.

The aim is to develop new heuristics based on PNs to solve SALBP-1. Reachability analysis, firing sequences of transitions (*T*-invariants), and *P*-invariants of PNs will be used as search techniques. In other words, this problem will be solved by the help of token movement in the net.

In this thesis, five heuristic algorithms based on Petri nets are proposed to solve SALBP-1. To assign a task to a station, token condition information is used by the reachability analysis algorithm; a firing sequence list predetermined according to the order of transition firing is used by the algorithms based on the firing sequence of transitions; the P-invariants algorithm uses predetermined alternative routes found by P-invariants. Proposed algorithms are tested on benchmarking data sets, and their efficiency are compared with each other and with other algorithms.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In chapter two, types of assembly line balancing problem are defined, and a review of the assembly line balancing research is given. Overview of PNs, and a review of the studies on manufacturing applications of PNs are given in chapter three. For solving SALBP-1, five algorithms based on Petri nets are proposed in chapter four. This chapter is followed by testing of the proposed algorithms and comparison of them with existing methods in chapter five. Finally, concluding remarks are made and future research directions are outlined.

CHAPTER TWO
ASSEMBLY LINE BALANCING

2.1 Terminology

Before the introduction of assembly line balancing problem, a terminology is given in follow.

Task: Job, which is done in assembly, line can be divided into its tasks. When all tasks are allocated to the workstations a feasible solution will be obtained.
Task time: Each task is completed in its task time.
Station time: Station time (or workstation time) is equal to total time of the tasks allocated in the workstation. Each task assignment process updates the workstation time by adding the time of the new assigned task to time of the previous assigned tasks.
Cycle time: Maximum amount of time the job (or product) is allowed to spend at each station if the targeted production rate is to be reached. Cycle time must not exceed the station time, and cycle time must not be less than maximum task time on the assembly line.
Precedence constraints: There may be technological or/and physical sequencing requirements on the assembly line. In other words, sometimes a task will not be done if some tasks are not completed.
Precedence diagram: A graphical representation of the sequence of tasks as defined by the precedence constraints.
Precedence matrix: A matrix has number of rows and columns as number of tasks on the line. If there is a precedence relation between i_{th} task and j_{th} task and i_{th} task must be done before the j_{th} task, then i_{th} row and j_{th} column of the matrix is equal to "+1" and j_{th} row and i_{th} column of the matrix is equal to "-1".

Balance delay: A measure of the line efficiency which results from idle time due to imperfect allocation of task among stations. It is computed as follows:

$$Balance\ \ delay = \frac{Number\ of\ stations\ in\ the\ line \times Cycle\ time - Total\ of\ station\ times}{Number\ of\ stations\ in\ the\ line \times Cycle\ time}$$

Flexibity-ratio (F-ratio): It is a measure of the number of feasible sequences that could be generated from an N-task problem, and can be expressed as follows:

$$F - ratio = \frac{2Y}{N \times (N-1)}$$

where Y is the number of zeros in the precedence matrix, (Erel & Sarin, 1998)
Order Strength: It measures the volume of distinct orderings that are permitted by the specified precedence diagram. Its formulation as follows (Erel & Sarin, 1998):

$$Order\ strength = \frac{2 \times (Number\ of\ \ precedence\ realtions)}{N \times (N-1)}$$

West Ratio: It gives the ratio of average number of tasks assigned to stations, and is expressed as follows (Erel & Sarin, 1998):

$$WEST - ratio = \frac{Number\ of\ tasks}{Number\ of\ stations}$$

2.2 Introduction

One of the concerns of companies to select a production system is to maximize production rate or to minimize production cost. Assembly system is a type of production system, and aims to maximize the production rate by minimizing the cycle time. Assembly systems include many stations on the line. When any task is completed at any station, product (or products) is (are) sent to one of the following stations on the line to complete another task. Important disadvantages of assembly systems are starving and/or blocking of stations on the line. Therefore, assembly line balancing has been a worthy problem for researchers from both academia and industry.

This chapter is organized as follows. Types of assembly systems will be introduced in the next section. Section 2.4 includes descriptions and types of assembly line balancing problems, and solution approaches. A literature review on types of assembly line balancing problems will be given in section 2.5. This chapter will be concluded with section 2.6.

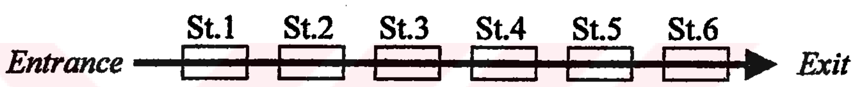
2.3 Assembly Lines

An assembly line is a set of sequential workstations, typically connected by a continuous material handling system. The line is designed to assemble component parts and to perform any related operations necessary to produce a finished product. The complete assembly activity of products is divided into productive work elements. Each workstation is assigned a subset of these work elements. The product is passed down the line, visiting each station in sequence. Upon existing the final station, the product is complete. The line is operated in such a way that the stations are simultaneously busy. Upon completion of its assigned tasks on an item (unit of product), the station passes the item to the next station, obtains a new item from its predecessor station, and repeats its tasks. (Askin & Standridge, 1993, p.31)

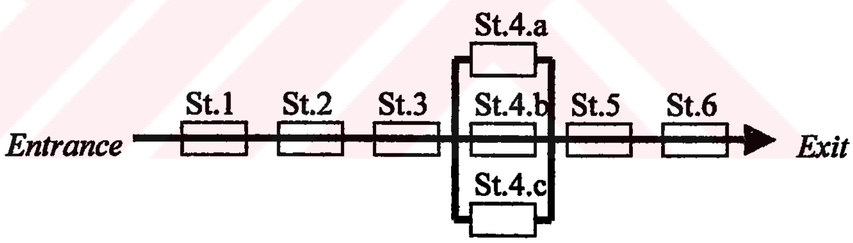
Assembly lines increase the production volume and decrease the production cost. First assembly line was applied in 1913 by Henry Ford. *Model T* is one of the first automobiles, and it was produced in 12 man-hours and its price was $ 850 before the assembly line application (1908). Assembly line production decreased production time to 2.5 man-hours thus its price decreased to $ 290. Assembly lines provide division of labor. Division of labor embraces the concepts of work simplification, standardization, and specialization (Askin & Standridge, 1993, p.31).

First application and simplest assembly line is the serial assembly line. A typical serial assembly line (or straight-line system) is given in figure 2.1.a. In this figure, several cells are arranged in series and typically connected by a continuous material handling system. Advantages of serial assembly line are: (Vairaktarakis & Szmerekovsky, 2001)

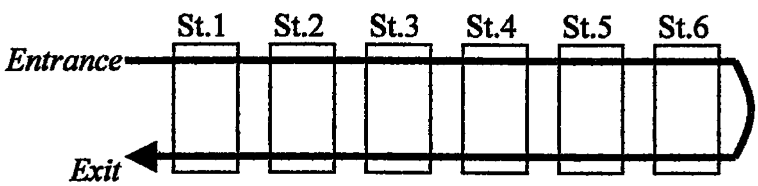
- Lower work-in-process inventories,
- Reduced material handling costs,
- Shorter production flow times,
- Simplified production planning of materials and labor,
- Improved visual control,
- Fewer tooling changes.



(a). Serial assembly line



(b). Assembly line with parallel stations



(c). U-type assembly line

Figure 2.1 Configurations of assembly lines

In some instances technology may be the deciding factor on configurations of assembly lines. Machines performing at the speed required to meet demand may not exist or may be prohibitively expensive. Major disadvantage of the serial assembly line is that when the any station on the line fails the production line will stop if any work-in-process inventories are not kept. Parallel stations are designed on the assembly line to remove this undesired condition. An assembly line with parallel stations is shown in figure 2.1.b. In this figure, three stations are designed in parallel in station 4. Therefore any fail in any of three stations doesn't stop the entire line, because other parallel stations complete tasks and send product to following stations for new tasks. Other advantages of assembly line with parallel stations are; (Askin & Standridge, 1993, p.34)

- Easier to balance work load between stations,
- Increased scheduling flexibility,
- Job enrichment,
- Higher line availability-worker independence,
- Increased accountability.

Assembly line with parallel stations has disadvantages as well (Askin & Standridge, 1993, p.34). Additional equipment or additional works may raise cost per unit. Setup cost and equipment cost increase. Learning occurs more slowly, since workers are not repeating the same activity as frequently. Therefore workers have high skill requirements. Also visual control of the line is more complex.

In 1993, Monden identified more general configurations (U- and S-type topologies) of the traditional straight-line system. An U-type configuration of assembly line is given in figure 2.1.c. Monden observed these configurations in Japanese manufacturing companies. The important characteristic of these systems is that multi-skilled (cross-trained) workers perform various tasks of different stations along the production line. This feature allows assigning tasks both from the beginning and the end of the precedence diagram to the same stations. Workers in a U-type configuration may work on two different items within the same cycle. The need for U-type assembly configuration arises from attempts to improve productivity with greater flexibility. Monden (1993) discusses these issues in the context of the

Toyota Production System. General advantages of the U-type configuration stated in Erel *et al.*, 2001 are

- A quick response to changes in the environment (machine breakdowns, worker absenteeism, etc.) due to the possibility of reallocating the cross-trained workers,
- The ease to adopt to changes in cycle time, because of the high potential of rebalancing the line with a new cycle time,
- A high level of participation of workers to improve production process,
- The flexibility of adding/removing workers (rebalancing lines), and, most importantly,
- The number of stations required by a U-line is never greater than the one by a traditional line.

In spite of these benefits, U-lines present some operational difficulties in scheduling the movements of workers, dispatching jobs, material handling activities and WIP control (Erel *et al.*, 2001).

All of the assembly line configurations have the same problem: how tasks are assigned to station. This problem is called line balancing problem and it will be explained in the next section.

2.4 Line Balancing Problem

The line balancing problem is to arrange the individual processing and assembly tasks at the workstations so that the total time required at each workstation is approximately the same. If the work elements were grouped so that all the station times to be exactly equal, the line balance would be perfect and the production flow would be smooth. In most practical situations it is very difficult to achieve perfect balance. When the workstations' times are unequal, the slowest station determines the overall production rate of the line (Groover, 1992, p.143).

Assembly line balancing problem is mainly classified into two classes according to the number of job (or product) on the line. If only one job are done on the line then problem is called as single assembly line balancing problem. If more than one job is done simultaneously on the line then problem is called as mixed assembly line balancing problem. Task times are also used for classification of the problem. If task times are constant then the problem is called as deterministic assembly line balancing problem. The deterministic assembly line balancing problem can be stated as follows: given a finite set of tasks, each having a fixed performance time, and a set of precedence relations which specify the permissible orderings of the tasks, the problem is to assign the tasks to an ordered sequence of stations such that the precedence relations are satisfied and some measure of performance is optimized (Erel & Sarin, 1998). If the task times have a randomly distribution then the problem is called as stochastic assembly line balancing problem. The stochastic assembly line balancing problem can be stated as follows: given a finite set of tasks, each having a performance time distributed according to a probability distribution and a set of precedence relations which specify the permissible orderings of the tasks, the problem is to assign the tasks to an ordered sequence of stations such that the precedence relations are satisfied and some measure of performance is optimized (Erel & Sarin, 1998). If the new classification is added into previous classification then problem is classified into four class; deterministic single assembly line balancing problem, stochastic single assembly line balancing problem, deterministic mixed assembly line balancing problem, and stochastic mixed assembly line balancing problem. Deterministic single assembly line balancing problem is focus in this thesis. In the literature, this problem is called as simple assembly line balancing problem (SALBP).

Baybars (1986a) defines SALBP by means of the following assumptions:

- All input parameters are known with certainty
- A task cannot be split among two or more stations
- Tasks cannot be processed in arbitrary sequences due to the technological precedence requirements
- All tasks must be processed
- All stations under consideration are equipped and manned to process any one of the tasks (i.e., it is assumed, in effect, that the fixed and variable costs associated

with all the stations are the same and, therefore, they need mot to be considered in the model)

- The task process times are independent of the station at which they are performed and of the preceding or following tasks (i.e., process times are fixed and, furthermore, they are not sequence dependent)
- Any task can be processed at any station (i.e., there are no positional, layout or zoning restrictions)
- The total line is considered to be serial with no feeder or parallel subassembly lines (and, therefore, process times are additive at any station) or any possible interaction of this type is ignored
- The assembly system is assumed to be designed for a unique model of a single product

Under this assumptions a typical SALBP is classified in three groups according to the objectives:

- *SALBP-1:* the objective is to minimize the number of stations on the line for given cycle time
- *SALBP-2:* the objective is to minimize the cycle time for given number of stations on the line
- *SALBP-E:* the objective is to maximize the line efficiency for cycle time and number of stations being variable.

The first two problems were defined by Baybars, (1986a), and last one was defined by Scholl in 1995 (Corominas, 1999).

Types of SALBP can be combined in U-type assembly line configuration as:

- *UALBP-1:* the objective is to minimize the number of stations on the line for given cycle time
- *UALBP-2:* the objective is to minimize the cycle time for given number of stations on the line
- *UALBP-E:* the objective is to maximize the line efficiency for cycle time and number of stations being variable.

In terms of the solution of the line-balancing problem, this implies that the solution of the U-type configuration dominates the solution of the traditional straight-line configuration due to the reduced precedence relations (i.e. the number of stations on a U-type configuration is less than or equal to the number required on a straight line). On the other hand, the problem of assigning tasks on a U-type configuration is more complex than a straight line due to the much larger search space. Moreover, the line-balancing problem of an U-line is much more complicated than the traditional line due to the increased search space. (Erel *et al.*,2001)

A SALBP-1 can be formulated as an integer programming (Palekar, 1998). Suppose that there are N tasks in the assembly process and assembly line must have a cycle time C. Further, let t_j represent the time required for task j. The precedence graph is represented in the form of a directed graph $G=(V,E)$ where the vertex set V represents the tasks and the edge set E represents immediate precedence between tasks, and P_j denotes the set of immediate predecessors of task j. P_j can be obtained from graph G.

For a feasible solution, t_j must be less than or equal to C for every task j. This also implies that in the worst case number of N stations, one station for each task, will be suffice. To model the problem assume that an assembly line has a number of N stations and tasks have to be assigned to these stations. The objective is to use as small as possible number of stations. Accordingly, following variables are defined:

$$x_{jk} = \begin{cases} 1 & \text{if task } j \text{ is assigned to station k;} \\ 0 & \text{otherwise.} \end{cases}$$

Constraints

Following constraints must be satisfied by any assignment of values to the variables so that the assignment represents a valid solution to the assembly line problem.

1. To ensure that each task is assigned to exactly one of the N workstations, following constraint is used:

$$\sum_{k=1}^{N} x_{jk} = 1 \qquad \text{for every task } j = 1, \ldots, N.$$

2. The available time at each workstation is restricted to the cycle time. Accordingly the time assigned to a workstation must be constrained as follows:

$$\sum_{j=1}^{N} t_j x_{jk} \leq C \qquad \text{for every station } k = 1, \ldots, N.$$

where t_j is the time required for task j

3. Precedence constraints given in graph G must be also enforced. If a task j_1 precedes task j_2 (i.e., $j_2 \in p_{j_1}$) and if task j_2 is assigned to station k_1, then task j_1 must be assigned either to station k_1 or to some station k<k_1. This can be represented by a following constraint

$$x_{j_2k_1} \leq \sum_{k=1}^{k_1} x_{j_1k} \qquad \text{for every station } k_1 = 1, \ldots, N \text{ and every } j_1 \in p_{j_2}; \text{ and } j_2 = 1, \ldots, N.$$

4. Finally, the usual integrality restrictions are applied:

$$x_{jk} \in \{0, 1\} \qquad \text{for all } j, k = 1, \ldots, N.$$

Objective function

Objective of the SALBP-1 is to minimize the number of workstations required. However, to formulate the problem it is assumed that a fixed number (N) of workstations is needed. To find the minimum number required, one way is to use lower number workstations first. This is achieved by creating a fictitious cost of assigning a task to a workstation. The cost for a given workstation is the same regardless of the task but varies with the workstation. To force the use of lower numbered workstations, a cost structure is designed which increases as the station number increases. One possibility is to use costs as follows:

$$c_{jk} = c_k \qquad \text{for all } j;$$

$$Nc_k \leq c_{k+1}.$$

where c is a cost (or penalty) value.

This ensures that it is cheaper to assign all tasks to a lower numbered station before a higher numbered station is used.

With a cost structure as defined above, the objective function can be written as follow:

$$\min \quad \sum_{j=1}^{N}\sum_{k=1}^{N} c_{jk}\, x_{jk}$$

Task restrictions

The model can be augmented to include other factors. In some cases, it is necessary that a particular pair of tasks should not be done at the same workstation. Such restrictions are required because of fire hazards, vibration problems etc. In such cases additional constraints must be added to the model. Suppose that tasks j_1 and j_2 must be separated. The appropriate constraint to be added is

$$x_{j_1k} + x_{j_2k} \le 1 \qquad \text{for all } k = 1, \dots, N.$$

Similarly, two tasks may be forced to be done at the same workstation. This is often necessary because of the use of expensive tooling or hazardous chemicals etc. While it is possible to mathematically model such restrictions, it is preferable to just combine the tasks into a single task. In such cases it may be possible to combine other intervening tasks into the combined task as well.

SALBP-1 is computationally difficult. Let consider a special case of this problem. In this special case, assume that there is no precedence. The resulting problem reduces to packing the tasks into the fewest number of workstations. This is the well-studied "bin-packing problem". The bin-packing problem belongs to a class of problems which are known to be computationally difficult (NP-Hard problems). While it can not be claimed that there does not exist an easy (more formally, polynomial) method to solve these problems, it is widely believed that this is indeed true. The reason for this belief is that the bin-packing problem belongs to a class of problems which are all related to each other in such a way that if any one of these problems is shown to be easy then all of these problems can be shown to be easy.

Since these problems have been the subject of study for several decades without yielding an easy method, it is widely believed that no such method exists. Since the bin-packing problem is a hard problem, the assembly line balancing problem must also be hard (Palekar, 1998).

Many researches have been done to solve SALBP-1. There have been a number of attempts to optimally solve the SALBP-1 using linear programming (LP) (Salveson, 1955), integer programming (IP) (Bowman, 1960; Klein, 1963; Patterson & Albracht, 1975; Talbot & Patterson, 1984), dynamic programming (DP) (Jackson, 1956; Held *et al.*, 1963; Schrage & Baker, 1978), and branch- and-bound (B&B) approaches (Jackson, 1956; Johnson, 1981; Wee & Magazine 1981). Since the optimal solution of even a modest size problem (e.g., with 100 tasks) is impossible by the exact methods, a considerable research effort has been spent to develop heuristic approaches. Among them, most notable ones are: MALB (Dar-El, 1973), MUST (Dar-El & Rubinovitch, 1979), LBHA (Baybars, 1986b), Tonge's (1965), Moodie & Young's (1965), and Nevins' (1972) heuristics. Baybars (1986b) compares his heuristic with Tonge's, Moodie & Young's, and Nevins' heuristics on Tonge's problems (Sabuncuoglu *et al.*, 2000).

A literature review about SALBP-1 and other assembly line balancing problems will be given in the next section.

2.5 Literature Review

Many solution approaches have been proposed (Sabuncuoglu *et al.*, 2000) since the assembly line balancing problem was first formulated by Helgeson *et al.* in 1954. In this section, literature review is not limited to only SALBP-1. Because, procedures, methodologies, and/or approaches, which are used for any line balancing problem, can be used (combined) for other line balancing problems. For example, Scholl & Klein's branch and bound procedure, SALOME, was firstly used for SALBP-1, and then SALOME-2 was developed for SALBP-2, and then ULINO was developed for UALBP by using procedures of SALOME. Another reason is that a type of assembly line balancing problem can be solved by transforming that type of problem into another type of assembly line balancing problem. For example, many researchers transformed mixed assembly line balancing problem into simple assembly line balancing problem by combining the precedence graphs.

Before literature review, general solutions approach (Erel & Sarin, 1998) to solve different types of assembly line balancing problem are given in figure 2.2.

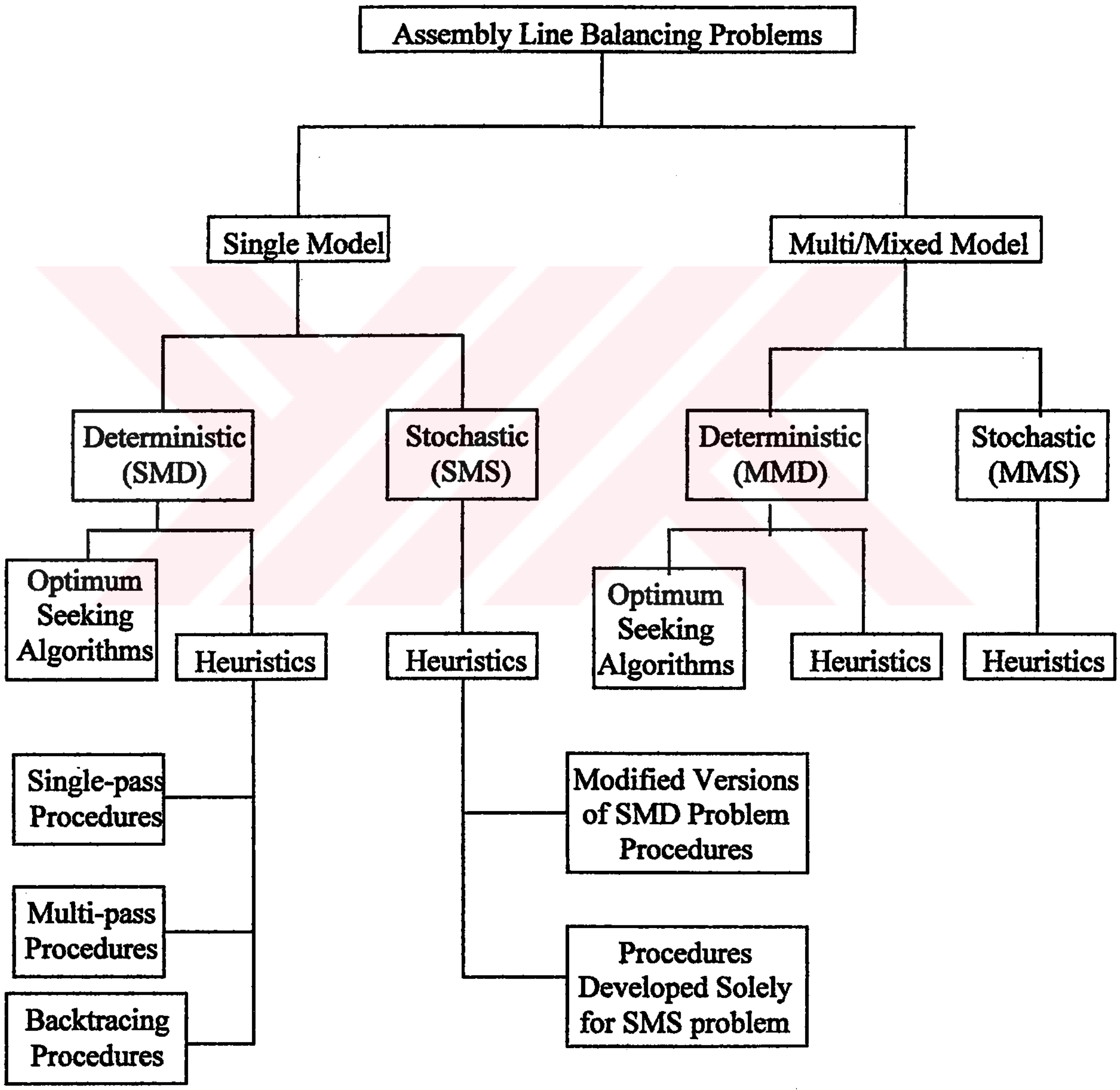


Figure 2.2 Classification of the assembly line balancing problem and the related solution procedures

As shown in figure 2.2, there are mainly four types of assembly line balancing problems. Different solution procedures have been used to solve any type of problem by researchers. To solve a deterministic assembly line balancing problem with single model two approaches have been used (Baybars, 1986a). In the first approach, optimal solution is achieved using optimum seeking algorithms based on integer programming, 0-1 programming or dynamic programming. Second approach includes the heuristic procedures, which reduce the size of the searching, but an optimal solution may not be found. Heuristic procedures are classified into three subclasses. The first category consists of single-pass decision rule procedures which implement a list of processing prioritizing scheme for task assignment based on a single attribute of each task. For example, Baybars (1986b) developed a heuristic based on single-pass decision. The second category consists of procedures that produce multiple single-pass solutions and select the most attractive solution. The last category consists of procedures that attempt to improve a solution or a station assignment by some iterative backtracking methods (Erel & Sarin, 1998). Talbot & Patterson (1984), Saltzman & Baybars (1987), Johnson (1988), Nourie & Venta (1991), and Hoffmann (1992) developed heuristics based on iterative backtracking methods. Also comparative studies have been done to evaluate the heuristics. For example Talbot *et al.* (1986) made a study to compare the single-pass, multiple single-pass, back tracing, and optimal seeking heuristics.

Only heuristics are used to solve stochastic assembly line balancing problem with single model. These heuristics are organized into two subclasses. In the first class, stochastic assembly line balancing problem is modified into deterministic type, and then problem is solved using the heuristics. In the second class, problem is solved using heuristics which are developed to solve only stochastic problems.

Deterministic assembly line balancing problems with mixed/multi model are solved by optimum seeking algorithms or heuristics (Erel & Sarin, 1998). In the stochastic type, problem is solved by heuristics which are based on a simulation study.

Literature review is organized into two sub sections: studies related single assembly line balancing problems and studies related mixed assembly line balancing problems.

2.5.1 Single Assembly Line Balancing Problem

If only one product is produced (or only one job is done) at the stations on the line the problem is called single assembly line balancing problem. There are many studies related with this problem type in the literature. A review only on recent studies is given in followings.

Rubinovitz & Levitin (1995) dealt with single assembly line balancing problem. Their solving procedure was based on genetic approach. All procedures (reordering, division, and exchange) of genetic approach, which were used in solving the assembly line balancing problem, were given by solving an example. They selected an experimental data set including three problems with 107 tasks and three problems with 142 tasks. Their F-ratio values, which are measure of the number of feasible sequences, were 0.1, 0.4, and 0.8. Each problem was solved for different number of stations such as 10, 15, 20, 25, 30, and 35. They made some tests to answer some questions like as "what is the effectiveness of exchange procedure"," does exchange procedure affect solutions", and etc. They also compared genetic approach results with the results of MUST (Multiple Solutions Technique), which was developed by Dar-El & Rubinovitch in 1971.

Study of Kim *et al.* (1996) has five objectives: minimizing the number of stations, minimizing the cycle time, maximizing the workload smoothness, maximizing the work relatedness, and maximizing last two objectives together. They used Genetic approach to solve single-model assembly line balancing problem (SALBP) with these objectives, and proposed a new decoding method using genetic approach for second and third objectives. The repair method, which is used to generate the genetic operators, was developed to select the efficient operator. For each objective, best operators were selected from nine operators tested. They compared their proposed genetic approach techniques with several existing heuristic procedures, which were minimum upper bound, maximum task time divided by task upper bound, maximum task time, largest set rule, and Rachamadugu & Talbot's method, using Kilbridge & Wester's 45-task and Tonge's 70-task problems. They found high improvement rate especially for third objective (%99). Other improvement rates were not very high. For example, these rates were %0-12.5 for first objective, %0-2.1 for second objective, and %3.3-10.8 for fourth objective. They also proposed a multi objective genetic approach for the fifth objective, and they used their proposed method for the solution of Tonge's 70-task problem.

Klein & Scholl (1996) interested in SALBP-2, and gave detailed information about existing methods for this problem. They developed a method called SALOME-2 to solve this problem. They proposed an enumeration procedure, the local lower bound method, and pointed out the similarities and differences between proposed and existing methods, such as FABLE (Johnson, 1988), and EUREKA (Hoffmann, 1992). They gave their insights on proposed method considering the solutions of problems with tasks, from 25 to 297, from two data sets with 128 and 174 instances, respectively. Totally 302 problems were solved using SALBP-1 based search methods, lower bound method, binary search, fibonacci and binary search, and binary search method with prespecified entry point. These problems were also solved using unidirectional and bidirectional versions of SALOME-2. All versions gave the more number of optimum solution than the all SALBP-1 based search methods. Finally, these problems were also solved using the existing procedures, FABLE, EUREKA, SALOME-1, and TBB (Task-oriented Branch and Bound procedure developed by Scholl in 1994). Results of the existing procedures were compared with those of flexible bidirectional version of SALOME-2. SALOME-2 gave the maximum number of optimum solution in this comparison.

Scholl & Voss (1996) focused on priority based methods for solving SALBP-1 and SALB-2. These methods produce an initial solution, and then this initial solution is improved. They proposed new priority rules, and illustrated the use of these rules in unidirectional and bidirectional procedures. These rules were also modified to be used as dynamic rules. Also they described a basic improvement procedure, which was combined with a static version of tabu search. The application of the procedure for solving SALBP-1 and SALBP-2 was given in the paper. Their test data set included totally 269 instances for SALBP-1 and 302 instances for SALBP-2. First test was performed for SALBP-1, and to compare unidirectional and bidirectional strategies according to the four static rules. Performance measures of this comparison were the number of optimum solutions, number of best solutions, average deviation (%), maximum deviation (%), average absolute deviation, and maximum deviation. Results were close each other, but bidirectional procedure gave the better results. Second test was performed for SALBP-2 using binary search methods and rules, and performance measures were the same. Again bidirectional procedure's results were good. Test problems were also solved using other three search methods, lower bound, Fibonacci binary, and binary search with prespecified entry point. Lower bound method gave the good results but their number of iterations and computation time were bad. Third test was made to solve SALBP-2 with tabu search using the test data (302 instances). Results were obtained according to the four different sizes of tabu list. Initial solutions were obtained using Fibonacci binary search method. Results showed that size of tabu list affects number of optimum solutions and best solutions. A part of the tabu list was used as dynamic list to improve the results. They again solved the set of problems using this idea in four different sizes of tabu list, and then small improvements in the results were observed. Also they tested influence of the initial feasible solution on the final solution. Problems were solved using Fibonacci binary search and four directional procedures such as backward, forward, bidirectional, and random. Random direction gave the worse results, and other results were close each other. However good final solution was obtained from initial solutions produced by bidirectional procedure. Also they showed that any conflict management strategy did not produce better results than the others. Last test were performed for SALBP-1 using tabu search. They used two different initial solutions for upper bound of number of stations; first solution was produced using priority rule, second was produced using EUREKA. Results obtained from two initial solutions were compared with the results of EUREKA and FABLE for three time limits; 50, 100 and 250 second. Tabu search with priority rule gave worse results than EUREKA and FABLE, but its performance was close to FABLE's for all time limits. However tabu search with priory rule found solutions more quickly than FABLE according to the average computation time. Results obtained in minimum average computation time by Tabu search with EUREKA were close to EUREKA's results and better than FABLE's.

Scholl & Klein (1997) proposed a new method, SALOME, which is based on branch and bound procedures, to solve simple assembly line balancing problem-type 1. In their paper, before SALOME was introduced, FABLE and EUREKA, which are most effective branch and bound procedures, were presented by giving the details of their branching and bounding processes. SALOME includes a new enumeration technique, which is called local lower-bound, bidirectional branching strategy, and a prefixing of tasks. Local lower-bound and its extensions run in branching and bounding stages, and reduces the search space. SALOME uses the maximum load rule, the task time incrementing rule, and an extension version of the Jackson dominance rule for logical test. To improve the searching area, SALOME uses bidirectional searches. They tested forward and bidirectional versions of SALOME on Talbot's, Hoffmann's, and Scholl's data sets. Their results were compared with FABLE's and EUREKA's results according to the performances measures, which are number of optimum solutions in percent, average relative deviation from optimal solution in percent, maximum relative deviation from optimal solution in percent, and average computation time. They conducted two tests for comparison. In the first test, forward procedures, which are FABLE, the forward procedure of EUREKA, and SALOME with forward direction, were compared. Time limit of computations was 500 seconds. SALOME gave best results in combined data set. Second test was applied on data sets using the bidirectional version of SALOME, FABLE, and EUREKA. Again SALOME gave better results than others for different time limits; 50, 100, 250, 500, and 1000 seconds. They made another comparison to examine the precedence relations and task times on the performance of the procedures using the combined data set. Two complexity measures were used in this comparison, such as order strength and time variability ratio, which shows the ratio between maximum task time in the problem and minimum task time (t_{max}/t_{min}). Test results were organized into three groups for order strength and into three groups for time variability ratio. Bidirectional version of SALOME produced best results for all classifications of order strength and time variability ratio.

Uğurdağ *et al.* (1997) developed a method based on integer programming for SALBP-2. This method equalizes the workload among workstations, and includes two steps; in the first step (STARTER), an initial solution is obtained using their heuristic procedure, and then initial solution is improved in the second step (ALMap) using a simplex like algorithm. Their assumptions are; tasks have constant time, all tasks must be processed, all workstations are identical, assembly line is serial, a task cannot be split, and precedence diagram cannot be ignored. The object is to minimize the cycle time. Constraints are; each task must be assigned to exactly one workstation, assignments must be done according to the precedence diagram, cycle time cannot be exceed in any station, and variables in the model must be binary. Obtaining initial solution step and improving the initial solution step were explained, and all steps of heuristic used in each step were given. They tested their method on Merten's 7-task, Jaeschke's 9-task, Jackson's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge & Wester's 45-task, Tonge's 70-task, and Arcus' 111-task problems. Six of 23 problems were solved optimally, but other results were near optimum. They also used 63 problems from Bowman's 9-task, Dar-Ell's 11-task, and Arcus' 83-task problems to test their proposed methods on, for workload smoothness. Their method gave better workload smoothness results than the workload smoothness results from optimum solutions, which are found for SALBP-2 by Wee & Magazine in 1981.

Ajenblit & Wainwright (1998) proposed a new genetic approach to solve UALBP-1. Firstly a feasible initial solution was generated, and then final solution was obtained from initial solution using a crossover operator, called ordered two-point crossover. They tested their method on 61 problems from Merten's 7-task, Bowman's 9-task, Jaeschke's 9-task, Jackson's 11-task, Dar-El's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge&Webster's 45-task, Tonge's 70-task, Arcus' 83-task and Arcus' 111-task problems. They used three versions of their algorithm to solve these problems. First version was for total idle time, the second was for balance of the workload among stations, and the third was a combination of previous versions. The results were compared with those of Dynamic Programming (DP) and best known heuristic algorithms. Forty of test problems with large size (from 21 tasks to 111 tasks) were not solved by DP since DP was too costly technique for these problems. Comparison was made with only heuristic results for other problems. Heuristics and DP have the same optimum solutions for first 21 problems. For 49 problems, Ajenblit & Wainwright's heuristic gave the similar results with the best results of other heuristics. Their heuristic was superior to other heuristics for 11 problems although it produced the worse result than other heuristics for one problem.

Chiang (1998) used tabu search methodology to solve SALBP-1. Tabu search is an improvement methodology, and improves the initial solution until it finds optimum, near optimum or acceptable solution by using neighborhood search techniques. Application of neighborhood techniques in assembly line balancing problem must be in coordination with precedence relations of tasks. For example movement of tasks between stations must be done using precedence diagram. Chiang developed four different assembly line balancing algorithms using tabu search methodology; first one uses best improvement with task aggregation, the second one uses best improvement without task aggregation, the third one uses first improvement with task aggregation, and the last one uses first improvement without task aggregation. First improvement search selects the first better solution than current neighborhood. Best improvement was a search technique that selects the best solution from the all neighborhood searches. Task aggregation was used to reduce the search space therefore an improvement mechanism (λ-exchange), which limits the number of task movements between stations, was adopted in tabu search strategy. A data set with 64 instances, which includes Merten's 7-task, Bowman's 9-task, Jaeschke's 9-task, Jackson's 11-task, Dar-El's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge&Webster's 45-task, Tonge's 70-task, Arcus' 83-task and Arcus' 111-task problems, were used to test his algorithms. He selected the tabu size for each problem as m/4, and maximal number of iteration as 50m, where m is the problem size. Solution results, number of stations, and CPU times were given for different cycle times, and these results were also compared with optimal solutions. Versions using best improvements found more number of optimal solutions but greater CPU times than those using first improvements.

Kim *et al.* (1998) developed a heuristic-based genetic approach (HGA) for workload smoothing, and tested it on Kilbridge & Wester's 45-task, Tonge's 70-task, Arcus's 83- and 111-task, and Bartholdi's 148 task problems. They firstly generated control parameters, and then tested them for the problem sets. Results were compared with those of three heuristics; maximum duration rule, modified Hoffmann procedure, and Rachamadugu & Talbot's procedure. Their approach outperformed for 24 of 31 problems.

Corominas (1999) interested with efficiency of an assembly line using SALBP assumptions. Mixed integer linear programming (MILP) was selected to solve this problem. Objective was to minimize the product of the cycle time by the number of open stations. Constraints were; every task must be assigned to a station, cycle time is not exceeded during the assignment of tasks to stations, station must be open for any assignment of task, precedence constraints are corresponded, open stations must be contiguous, and corresponding variables must be binary. MILP model was also given for SALBP-2. As lower bound, optimum value of objective function was used to decrease the time and solution space, and therefore Dillenberger's procedure was used by this aim. Using MILP, he solved problems with 20, 30, 40, and 60-task, for SALBP-2 and SALBP-E. Results were found using LINDO software package. Five of seven SALBP-2 problems, and four of seven SALBP-E problems were solved optimally. Maximum deviation from the optimum was %2.33 in no optimal results. All SALBP-2 problems were solved at most one hour whereas solution time of three SALBP-E problems exceeded two hours.

Scholl & Klein (1999a) made a comparison among most effective branch and bound procedures for SALPB1, such as Johnson's FABLE, Nourie & Venta's OptPack, Hoffmann's Eureka, and Scholl & Klein's SALOME. All procedures were explained according to some properties like branching (enumeration), bounding, and logical tests. They used Talbot's data set, Hoffmann's data set, and Scholl's data set, which are includes 269 instances totally, for comparison of performance of all procedures. Performance measures were average relative deviation from optimality, maximum relative deviation from optimality, average computation time, and average rank. The results of comparison study were organized into three class according to the data set. In the Talbot's data set, SALOME was the most effective procedure. FABLE, OptPack, and Eureka followed SALOME respectively. In the Hoffman's data set, OptPack was the most effective procedure, SALOME was second, FABLE was third, and Eureka was last. SALOME was again most effective procedure in Scholl data set. All of the other procedures were second. SALOME was determined as a most effective branch and bound procedure in the study. However other procedures had got some superior properties, for example OptPack was very effective in reducing the size of the enumeration tree. Therefore SALOME was improved by adding tree dominance rule and other superior properties. New version outperformed previous version of SALOME for all data sets.

Scholl & Klein (1999b) improved the heuristic (ULINO), which was derived from SALOME introduced in their previous study (Scholl & Klein, 1997), for U-line assembly systems. ULINO is based on branch and bound algorithm. Branching, bounding, and logical testing steps of algorithm were explained by solving an example problem with 12 tasks. Applications of ULINO on SALBP-1 (UALPB-1), SALBP-2 (UALBP-2) and SALBP-E (UALBP-E) were given in this study. They performed their heuristic on a problem set for UALBP-1 (269 instances), UALBP-2 (302 instances), and UALBP-E (256 instances). Results include number of optimum solutions, average deviation from optimum, maximum deviation from optimum, and average CPU time. All tests were performed for time limits with 500 second. 233 of 269 problems were solved optimally for UALBP-1. Maximum and average deviations from optimum were 10, and 0.59, respectively. They also compared optimum solutions of SALBP-1 with those of UALBP-1 for 154 problems. ULINO decreased the average deviation from 11.66 (for SALBP-1) to 7.90 (for UALBP-1). For SALBP-2, three search procedures were used in ULINO. ULINO with lower bound search solved 258 of 302 problems optimally. With binary, and Fibonacci binary searches it achieved 247 and 254 optimum solutions, respectively. However binary search gave less average deviation (0.31) and less maximum deviation (9.09) than other search methods. These values were 6.87 and 89.45 for lower bound search, 6.53 and 89.45 for Fibonacci binary search. They also compared optimum results of ULINO with those of SALB-2. Except Binary search method, search methods gave higher average deviation than SALBP-2. ULINO solved 223 of 256 problems optimally for UALBP-E. When ULINO solutions were compared with optimal solutions of SALBP-E, ULINO gave higher average and maximum deviations than SALBP-E solutions. When the comparison was made for only 223 optimum solutions, ULINO gave better average deviation then SALBP-E and same maximum deviation with SALBP-E.

Sprecher (1999) developed a branch and bound algorithm, adapted general sequencing algorithm (AGSA), based on the resource-constrained project scheduling problem for SALBP-1. He formulated SALBP-1 problem as a single-mode resource-constrained project scheduling problem by reflecting cycle time as a single renewable resource. He tested his algorithm on three problem sets; on Talbot with 64 instances, on Hoffmann with 50 instances, and on Scholl with 168 instances, and compared results with the results of other branch and bound algorithms; EUREKA, FABLE, and SALOME. Performance criteria were the number of optimum solutions, average and maximum deviations from optimum, and average CPU times in second. When the time limit was 500 s., proposed algorithm solved 224 of 269 problems, optimally. This number was bigger than the number of optimum solutions obtained by using other branch and bound procedures. He also tested AGSA on the same problems but for different time limits; 50 s., 100 s., 250 s., and 1000 s. AGSA again gave better results for all performance measures and for all time limits.

Bautista *et al.* (2000) were interested with SALBP-1 by considering incompatibilities between groups of tasks. That is, if two tasks are incompatible they cannot be assigned to the same workstation. Bautista *et al.* developed a new approach based on greedy randomized adaptive search procedure (GRASP) obtained from the application of some classical heuristics. These heuristics were based on priority rules. They also used genetic approach to search for the solution in the heuristic space. They proposed greedy randomize weighted adaptive search procedure (GRWASP) by improving the GRASP. They made a computational analysis using 160 instances, which are divided into 4 groups of 40 instances with 20, 40, 60, and 80 tasks. Six versions of GRWASP were used to solve these problems, and results were compared with those of 13 greedy heuristics, 6 GRASP heuristics, and 7 versions of genetic algorithm which they presented. Genetic algorithms produced best results.

GRWASP's results were better than the all greedy heuristics and some of the GRASP heuristics.

Sabuncuoglu *et al.* (2000) proposed a new genetic approach to solve the SALBP-1. This approach uses Dynamic Partitioning (DPA), which developed by them and modifies chromosome structures of genetic approaches to save CPU time. They solved 30 different problems to investigate the effectiveness of DPA. They tested the effects of different DPA and genetic approach parameters (Dynamic-Partitioning-Constant with four levels, warm-up period with five levels, number of iterations with three levels, and population size with four levels). 30 problems with 50 tasks were randomly created for three F-ratios; 10 %, 50%, and 90%. Experiment was made with 10 replications for each problem at each combination of factor levels by using the same set of 10 random seeds. Therefore, 72000 problems were solved. Effects of all factors on CPU times and fitness score were tested using ANOVA. They used Kilbridge & Wester's 45-task problem to compare their proposed genetic approach method with existing genetic approach method developed by Leu, Matheson, and Rees in 1994, and five non-genetic heuristics; Moodie & Young's heuristic, MIF, Tonge's RC, BPC, and Nevins' heuristic. Performance measures of the comparison were mean squared idle time, square root of mean squared idle time, efficiency (utilization), and maximum station time. Their proposed method outperformed others. Using Tonge's 70-task problem as a test data for 13 different cycle times they made another comparison with LBHA-1 (Baybars 1986b), and the five heuristics. Their approach outperformed others except Nevins' and Baybars' heuristics.

Erel *et al.* (2001) developed a heuristic based on simulated annealing to solve the balancing problem of U-type assembly lines. Their heuristic consists of two parts. In the first part, a solution generator module creates new (better) solution from the old solution by relaxing the cycle time constraints, and then simulated annealing makes feasible assignments to maximize throughput rate using the solution created in the first part. Jackson's 11-task problem was solved using their algorithm, and all iterations of solution were given in the paper. Their heuristic was also tested on two data sets: first one is Miltenburg and Wijngaard's, Urban's, and Hoffmann's data set, and second one is Scholl & Klein's data set. They classifed the first data set into three categories: small (up to 11 tasks), medium (up to 45 tasks), and large (above 45 tasks). Optimum solutions for small and medium sizes are available in the literature. They found optimum solutions for 20 instances of 21 small size problems, and for 23 instances of 23 medium size problems. However, Ranked Positional Weight Procedure (RPWP) yielded 17 and 11 optimal solutions for small size and medium size problems, respectively. Optimum U-line solutions for large size problems were not available in the literature. Therefore when they compared their procedure with RPWP, two procedure gave the same results. They compared their heuristic with Scholl & Klein's ULINO using second data set including 168 problems with tasks from 25 to 297. 125 of 168 problems were solved optimally. Both algorithms produced the same results for 155 problems. Their proposed heuristic produced better results than ULINO for 8 of remain 13 problems.

2.5.2 Mixed Assembly Line Balancing Problem

Mixed assembly line is a production line with a variety of product models having similar assembly characteristics. If more than one product model are simultaneously produced (or more than one job are done) at the stations on a line, the problem is called mixed assembly line balancing problem. This problem involves assignment and sequencing. In the single assembly line balancing problem, the tasks are assigned to the stations, whereas in mixed assembly line balancing problem both tasks are assigned to stations on the line and production sequence of product models are determined. A review on recent mixed assembly line balancing studies is presented in this subsection.

Fokkert & Kok (1997) made a literature survey of mixed and multi model line balancing problems. Two approaches, the combined precedence diagram and the adjusted task processing times, were presented, and transformation of mixed and multi model problem to single model line balancing problem is explained. They designed a factorial experiment to compare the existing heuristics based on the combined precedence diagram, and to test factor effects. Factors in the experiment were number of tasks per model, position of common tasks in the precedence diagram, number of columns per model, model mix, and shift time on the performance measure. They also compared four balancing methods, which were Thomopoulos' two methods (one of them developed in 1967 other in1970), Macaskill's method (developed in 1972), Berger, Bourjolly, and Laporte's method (developed in1992), according to number of stations, unequal distribution of the total work content of the single models among the stations, and CPU time.

Gokcen & Erel (1997) presented a binary goal programming model for mixed-model assembly line balancing problem. Their assumptions were; similar models are produced on the same line, task times are known and constant, precedence diagram are known, WIP inventory is not allowed, common tasks of different models are assigned to the same stations, number of stations are the same for all models, and parallel stations are not allowed. An illustrative example with two models and with 10 tasks was given. After the precedence matrix was built, earliest and latest stations formulations were presented. Constraints were assignment constraints, which guarantee allocation of tasks of each model to at most one station, precedence constraints, cyclic time constraints, zoning constraints, which refer to the desirability and undesirability of assigning tasks into the same station, and station constraints, which provide each model to utilize the same number of stations. The goals were; number of stations should not exceed 3, cycle time should not exceed 22 and 24 for model 1, and for model 2, respectively, and task 1 and task 3 should be not be assigned to the same station. They used proposed goal programming model to this example problem. Model solution was found using GAMS, which is a software to solve linear programming models, by satisfying the first and third goals. Although goal sequence was changed and then model with six alternative goal sequences was solved, there wasn't any result satisfying the all goals. Also, they solved three two-model problems with 20, 30, and 40 tasks in the combined precedence diagram, respectively, using the proposed goal programming model. Each problem was solved three times; only first goal was used in the first problem, only first and second goals were used in the second, and finally all goals were used in the last.

Mullen & Frazier (1997) proposed a heuristic for solving mixed-model line balancing problems with stochastic tasks and parallel stations. Their heuristic was a modification of Gaither's heuristic for deterministic single-model proposed in 1996.

They improved this heuristic for stochastic mixed-model. Seven selected rules were used in heuristics to assign a task to a station. These rules were; task given highest cell utilization is selected, tasks are randomly selected (an application of COMSOAL), task having longest simulated task duration is selected (Moodie & Young's logic), task having shortest simulated task time is selected, task given lowest cell utilization is selected, task giving highest probability of completing all tasks in the cell on time is selected, and task giving highest cell utilization and has highest probability of completing all tasks in the cell on time is selected. Heuristic calculates the line balance statistics in the last step such as utilization of the overall line, and overall system probability of on-time completion. They designed a factorial experiment to evaluate the performance of the heuristic using 6 different problems, which were Tonge's 21-task, Buxey's 29-task, and Thomopoulos' 45-task problems, and randomly generated problems with 25, 40, and 74-task. Average WIP inventory level, average system flow time, units of throughput, average unit labor cost, percentage of units being completed within cycle time, average system utilization, and percent of desired cycle time attained were used as performance measures. They added three rules to presented seven rules. First added rule is lexicographic task selection rule, second one minimizes the number of workers but simultaneously maximizes the amount of necessary parallel stations, and third one minimizes the amount of necessary parallel stations but simultaneously maximizes the number of workers. Totally 60 layouts were built using each rule for each problem. Each of these layouts was simulated with 25 replications, and performance measures were calculated. The task durations were generated via a random number generator. They used MANOVA to search multivariate effect of rules selection on performance measures, effect of performance measures on rules selection, best selection rule given the best layout, and effect of number of work centers on the performance measures. They found that rules selection affects performance measures. They also showed that performance measures affect rule selection by making discriminant analysis and univariate ANOVA, and calculating means and standard deviations of performance measures for each rule selection. They obtained a regression model to test the effect of number of work centers on performance measures. As a result, more work centers increased average WIP level, throughput and on-time completion, while decreasing average system utilization and percentage of desired cycle time.

Erel & Gokcen (1999) interested with mixed model assembly line balancing problem. They combined models as a single assembly line model by assuming common tasks are processed in the same station. After that, shortest-route model, which was developed by Gutjahr & Nemhauser in 1964, was used to solve a line balancing problem including 9 tasks. Although 2^9-1=511 feasible states could be occurred by ignoring the precedence relations between tasks, they obtained a network containing 46 feasible states using precedence relations between tasks, and found optimal number of stations was four. When they used shortest-route formulation of Roberts & Villa developed in 1970, optimal number of stations was three for the same problem. Feasible states were reduced to 41 when constraints; cycle time, and processing common tasks in the same station were used. They used again Gutjahr & Nemhauser's shortest-route model on this network and found the same optimal solution.

Merengo *et al.* (1999) dealt with balancing and sequencing problems in a mixed-model assembly line. They presented new balancing and sequencing methodologies to achieve the objectives, which were minimizing the rate of incomplete jobs or the probability of blocking/starvation events, and reducing WIP. Also balancing methodology aims to minimize number of stations, and sequencing methodology provides a uniform part usage. They developed a balancing algorithm in four different versions according to the different combination of constraints and the horizontal balancing functions. Balancing algorithm includes three steps; an initial solution is searched using horizontal balancing in the first step, initial solution is improved by reducing the number of stations in the second step, and solution in the second step is improved by vertical balancing. Developed sequencing algorithm provided the minimum number of incomplete jobs. They tested their methods on 12 different problems with 40 tasks. Also totally 100 tests were applied on each problem, since task time was stochastic. Processing times and production volumes were randomly generated for each test. The number of stations found by their suggested methodologies was compared with the theoretical minimum number of stations. They also compared proposed sequencing method with Miltenburg's sequencing algorithm developed in 1989. Their methodologies outperformed Miltenburg's algorithm.

Kim *et al.* (2000a) dealt with line balancing and sequencing problems, which affect each other, in a mixed model assembly line, simultaneously. Their new methodology was based on co-evolutionary algorithm, which is a search technique simulating the evolutionary process in nature where one species has influences on and at the same time is influenced by other species. The aim was minimizing utility work, which is defined as the amount of uncompleted works within the given length of workstation. Balancing problem and sequencing problem were defined as populations. Generation, crossover and mutation operations, genetic representations, and adaptations of all of them to line balancing problem were explained in detail by representing proposed algorithm. They used Thomopoulos' 19-task and Arcus' 111-task problems and a problem with 61-task from real life to perform their algorithm. Results, improvements, and interpretations were given in the paper.

Kim *et al.* (2000b) proposed coevolutionary algorithm, a new approach based on an artificial intelligence technique, to solve model sequencing and line balancing problems in an U-line simultaneously. Thomopoulos' 19-task problem, Kim's 61-task problem, and Arcus's 111-task problem were selected as test-bed problems. Totally 21 problems were constructed so that 3 problems from Thomopoulos, 6 problems from Kim, and 12 problems from Arcus. They compared proposed co-evolutionary algorithm with hierarchical approach, which solves a sequencing problem after solution of line balancing problem, he tightly-coupled co-evolutionary algorithm, and loosely-coupled co-evolutionary algorithm according to the absolute deviation of workloads. Results showed that proposed algorithm outperforms others. Also it improved the results according to the hierarchical approach as from %28.20 to %73.20.

Recent studies on assembly line balancing problem were presented in previous two subsections. All studies are summarized in table 2.1 according to the year of the study, name of the author, problem type, solution approach, name of the developed algorithm, and test data sets. Areas of on-going research on ALBP are improving new search methods using genetic approach and coevolutionary algorithms, and are focusing on U-shape assembly lines.

Table 2.1 Studies in the Literature Review

Year	Researcher(s)	Problem Type	Solution Approach	Name of Dev. Alg.	Test Problems
1995	Rubinovitz & Levitin	SALBP	Genetic Approach	-	Randomly generated 6 problems (107-task problem and 142-task problem for F-ratio values of 0.1, 0.5,and 0.8) (no information about problems)
1996	Kim *et al.*	SALBP	Genetic Approach	-	Kilbridge & Wester' 45-task and Tonge's 70-task problems
1996	Klein & Scholl	SALBP-2	Branch and Bound Procedure	SALOME-2	Scholl's data set 1 with 128 instances Scholl's data set 2 with 174 instances
1996	Scholl & Voss	SALBP-1, -2	Tabu Search	-	269 instances for SALBP-1 (Talbot's data set, Hoffmann's data set, and Scholl's data set) 302 instances for SALBP-2 (Scholl's data set 1 and 2)
1997	Fokkert & Kok	Mixed	Heuristics	-	-
1997	Gokcen & Erel	Mixed	Binary Goal Programming	-	Two model problems with 10-task, 20-task, 30-task, and 40-task (no information about problems)
1997	Mullen & Fraizer	Mixed	Heuristic and Simulation		Tonge's 21-task, Buxey's 29-task, Thomopoulos's 45-task, and randomly generated problems with 25, 40, and 74-task
1997	Uğurdağ *et al.*	SALBP-2 and workload smoothness	Integer Programming	STARTER, ALMap	23 instances for SALBP-2 (Merten's 7-task, Jaeschke's 9-task, Jackson's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge & Wester's 45-task, Tonge's 70-task, and Arcus' 111-task problems) 63 instances for workload smoothness (Merten's 7-task, Jaeschke's 9-task, Bowman's 9-task, Dar-Ell's 11-task, , , Jackson's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge & Wester's 45-task, Tonge's 70-task, Arcus' 83-task and Arcus' 111-task problems)
1997	Scholl & Klein	SALBP-1	Branch and Bound Procedure	SALOME	269 instances (Talbot's data set, Hoffmann's data set, and Scholl's data set)

Table 2.1 (Cont.)

Year	Researcher(s)	Problem Type	Solution Approach	Name of Dev. Alg.	Test Problems
1998	Ajenblit & Wainwright	UALBP-1	Genetic Approach	-	61 instances (Merten's 7-task, Bowman's 9-task, Jaeschke's 9-task, Jackson's 11-task, Dar-El's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge&Webster's 45-task, Tonge's 70-task, Arcus' 83-task and Arcus' 111-task problems)
1998	Chiang	SALBP-1	Tabu Search	-	64 instances (Merten's 7-task, Bowman's 9-task, Jaeschke's 9-task, Jackson's 11-task, Dar-El's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge&Webster's 45-task, Tonge's 70-task, Arcus' 83-task and Arcus' 111-task problems)
1998	Kim *et al.*	SALBP (workload smoothness)	Genetic Approach	HGA	30 instances (Kilbridge & Wester's 45-task, Tonge's 70-task, Arcus's 83- and 111-task, and Bartholdi's 148 task problems)
1999	Corominas	SALBP-E	Mixed Integer Linear Programming	-	7 instances (20-task, 30-task, 40-task, and 60-task problems. no information about instances)
1999	Erel & Gokcen	Mixed	Shortest Route Model	-	Two model problem with 9-task (information was given)
1999	Merengo *et al.*	Mixed	Heuristic	BILVXOD	Problem with 40 tasks (information was given) (12 models)
1999	Scholl & Klein, (a)	SALBP-1	Branch and Bound Procedure		269 instances (Talbot's data set, Hoffmann's data set, and Scholl's data set)
1999	Scholl & Klein, (b)	UALBP-1,-2,-E	Branch and Bound Procedure	ULINO	269 instances for UALBP-1 (Talbot's data set, Hoffmann's data set, and Scholl's data set) 302 instances for UALBP-2 (Scholl's data set 1 and 2) 256 instances for UALBP-E 8Scholl's data set 3)

Table 2.1 (Cont.)

Year	Researcher(s)	Problem Type	Solution Approach	Name of Dev. Alg.	Test Problems
1999	Sprecher	SALBP-1	Branch and Bound Procedure	AGSA	269 instances (Talbot's data set, Hoffmann's data set, and Scholl's data set)
2000	Bautista *et al.*	SALBP-1	Genetic Approach and Heuristic	GRWASP	160 instances (40 instances with 20-task, instances with 40-task, instances with 60-task, instances with 80-task. no information about instances)
2000	Kim *et al.*(a)	Mixed	Coevolutionary Algorithm	PCoA, TCoA, and LCoA	Thomopoulos' 19-task and Arcus' 111-task problems and a problem 61-task from real life
2000	Kim *et al.*(b)	Mixed U-lines	Coevolutionary Algorithm	PCoA, TCoA, and LCoA	Thomopoulos' 19-task and Arcus' 111-task problems and a problem 61-task from real life
2000	Sabuncuoglu *et al.*	SALBP-1	Genetic Approach	-	Kilbridge & Wester's 45-task and Tonge's 70 task problems
2001	Erel *et al.*	UALBP-1	Simulated Annealing	-	Data set was created from earlier studies (Miltenburg & Wijngaard 1994, Urban 1998, Hoffmann 1990) 168 instances (Scholl's data set)

Notes related Table 2.1.

- *Talbot's data set* includes Bowman's 8-task, Mansoor's 11-task, Mertens' 7-task, Jaeschke's 9-task, Jackson's 11-task, Mitchell's 21-task, Heskiaoff's 28-task, Sawyer's 30-task, Kilbridge & Wester's 45-task ,Tonge's 70-task, Arcus' 81-task, Arcus' 111-task problems
- *Hoffmann's data set* includes Sawyer's 30-task, Kilbridge & Wester's 45-task ,Tonge's 70-task, Arcus' 81-task, Arcus' 111-task problems
- *Scholl's data set* includes Roszieg's 25-task ,Buxey's 29-task, Gunther's 35-task, Hahn's 53-task, Warnecke's 58-task, Wee-Mag's 27-task, Lutz's 32-task, Lutz's 89-task, Lutz's 89-task, Mukherje's 94-task, Barthold's 148-task, Bartholdi's 148-task, Scholl's 297-task problems
- *Scholl's data set 1* includes Buxey's 29-task, Gunther's 35-task, Kilbridge & Wester's 45-task, Lutz's 32-task, Lutz's 89-task, Sawyer's 30-task, Tonge's 70-task, Arcus' 83-task, Arcus' 111-task problems.
- *Scholl data set 2* includes Hahn's 53-task, Warnecke's 58-task, Wee-Mag's 27-task, Lutz's 89-task, Mukherje's 94-task, Barthold's 148-task, Barthold's 148-task, Scholl's 297-task problems
- *Scholl's data set 3* includes Mertens' 7-task, Bowman's 8-task, Jaeschke's 9-task, Jackson's 11-task, Mansoor's 11-task, Mitchell's 21-task, Roszieg's 25-task, Heskiaoff's 28-task, Buxey's 29-task, Sawyer's 30-task, Lutz's 32-task, Gunther's 35-task, Kilbridge & Wester's 45-task, Hahn's 53-task, Warnecke's 58-task, Tonge's 70-task, Arcus' 81-task, Lutz's 89-task, Lutz's 89-task, Arcus' 111-task, Bartholdi's 148-task, Bartholdi's 148-task, and Scholl's 297-task problems.

2.6 Conclusion

In this chapter, assembly line balancing problem, an important problem of assembly lines, types of this problem, and a literature review including the solution methods, and comparisons using several test data for some performance measures were presented. As an alternative to the other solution methods of SALBP-1, Petri net approach will be proposed in the following chapter.

CHAPTER THREE

PETRI NETS AND APPLICATIONS IN MANUFACTURING SYSTEMS

3.1 Introduction

Before the presenting alternative algorithms based on Petri nets to solve SALBP-1, Petri net information and its manufacturing applications will be shown in this chapter. General information on Petri nets will be given in Section 3.2. Section 3.3 includes manufacturing applications of Petri nets. A literature review, which consists of manufacturing applications of Petri nets, will be given in the last section.

3.2 Petri Nets

Petri nets are mathematical and graphical tools used for modeling, formal analysis, and design of discrete event systems. They allow for describing and studying systems which are characterized such as concurrent, asynchronous, distributed, parallel, non-deterministic, stochastic, and etc. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks (Murata, 1989). Tokens in a PN model provide to simulate the dynamic system. As a mathematical tool, it is possible to set up state equations, algebraic solutions, and other mathematical models governing the behavior of systems.

Historically speaking, the concept of the Petri net has its origin in Carl Adam Petri's dissertation, submitted in 1962 to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany (Murata,1989). He described, using a net, the causal relationships between events in a computer system. After his study, A.W. Holt and others illustrated how Petri nets could be used in modeling and analyzing systems of many concurrent components in between 1968-1971. Several dissertations and technical reports were published during the early 1970's in

Massachusetts Institute of Technology. In the late 70's, Europeans were very active in organizing workshops and publishing conference proceedings on Petri nets. These studies and next studies in following years have been published by Springer-Verlag as Advances in Petri nets in the series of Lecture Notes of Computer Science. Researchers with engineering background started their searches in the application of Petri nets in engineering systems particularly automated manufacturing systems in the early 80's. In the last two decays, Petri nets were used as a powerful tool in describing event-driven systems, which are asynchronous, contain sequential and concurrent operations, and involve conflicts, mutual exclusion and non-determinism (Peterson,1981; Murata,1989; Reisig, 1992; DiCesare *et al.*, 1993; Zhou & DiCesare,1993; Zurawski & Zhou,1994; Desrochers & Al-Jaar,1995; Moore & Gupta, 1996; Proth & Xie,1996; Wang,1998; Zhou & Venkatesh,1999).

3.2.1 Description of Petri Nets

A Petri net may be defined as a particular kind of bipartite directed graph populated by three types objects (Zurawski & Zhou, 1994). Objects are places, which are drawn as circle in graphical representation, transitions, which are drawn as box or bar, and directed arcs, which provide connection between transitions and places. There is not any connection between places or between transitions. A simple Petri net configuration is given figure 3.1. Places are classified in two types according to the direction of the arcs, which connect them to the transitions. For example, p_1 is input place of first transition (t_1), and p_2 is output place of t_1. p_2 is output places of t_2 and t_3, p_3 is output place of t_2, and so on. Output and input places represent some information related with modeled system. Some typical interpretations of transitions and places are given table 3.1. (Murata, 1989).

Tokens, which are pictured small solid dots in places, are very useful in order to study dynamic behavior of the modeled system. Tokens move between places according to the system properties. The movement or distribution of tokens on place is called Petri net marking, which defines the current state of the modeled system.

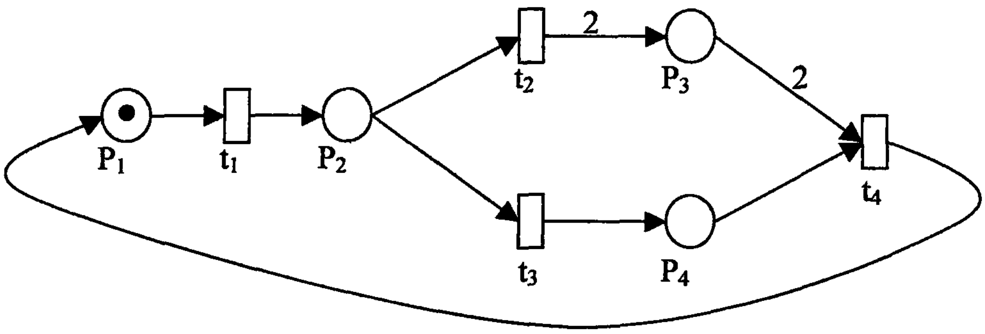


Figure 3.1 Petri net representation, an example

Table 3.1 Some Typical Interpretations of Transitions and Places

Input Places	Transitions	Output Places
Preconditions	Event	Postconditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signal
Resource needed	Task or job	Resources released
Conditions	Clause in logic	Conclusion(s)
Buffers	Processor	Buffers

The marking of Petri net model with m places is represented by an (mx1) vector M. M includes nonnegative integer numbers, and $M(p)$ represents the number of tokens in p place. M_0 is the initial token condition of Petri net model. For example M_0 for model given in figure 3.1 is [1 0 0 0]. Initial token condition represents the minimum number of tokens in required places to make a dynamic study with Petri net model.

Formally, a Petri net can be defined (Zurawski & Zhou, 1994) as follows:

PN=(P, T, I, O, M_0); where

1. $P = \{p_1, p_2, \ldots.., p_m\}$ is a finite set of places,

2. $T = \{t_1, t_2, \ldots.., t_n\}$ is a finite set of transitions, $P \cup T \neq \varnothing$, and $P \cap T = \varnothing$,

3. I: $(P \times T) \rightarrow N$ is an input function that defines directed arcs from places to transitions, where N is a set of nonnegative integers,

4. O: $(P \times T) \rightarrow N$ is an output function which defines directed arcs from transitions to places, and

5. $M_0 : P \rightarrow N$ initial marking.

$I(p,t) = k$ represents that there exist k parallel arcs connecting place p to transition t. Also $O(p,t) = k$ determines that there exists k parallel arcs connecting transition t to place p. If k is equal to 0 then there isn't any arc between places and transitions.

Properties of Petri net model change when tokens move from any place to other place. This movement or flow of tokens is based on rules (Zurawski & Zhou, 1994). They are;

Enabling Rule: A transition t is said to be enabled if each input place p of t contains at least the number of tokens equal to the weight of directed arc connecting p to t.

Firing Rule:

a) An enabled transition t may or may not fire depending on the additional interpretation, and

b) A firing of an enabled transition t removes from each input place p the number of tokens equal to the weight of the directed arc connecting p to t. It also deposits in each output place p the number of tokens equal to the weight of the directed arc connecting t to p.

The enabling and firing rules are represented in figure 3.2. (Zurawski & Zhou, 1994). In figure 3.2.(a), transition t_1 is the input place p_1 of transition t_1 contains two tokens, and $I(p_1,t_1)$=2. The firing of the enabled transition t_1 removes from the input place p_1 two tokens as $I(p_1,t_1)$=2, and replaced one token in the output place p_3, $O(p_3,t_1)$=1, and two tokens in the output place p_2, $O(p_2,t_1)$=2, this shown figure 3.2.(b).

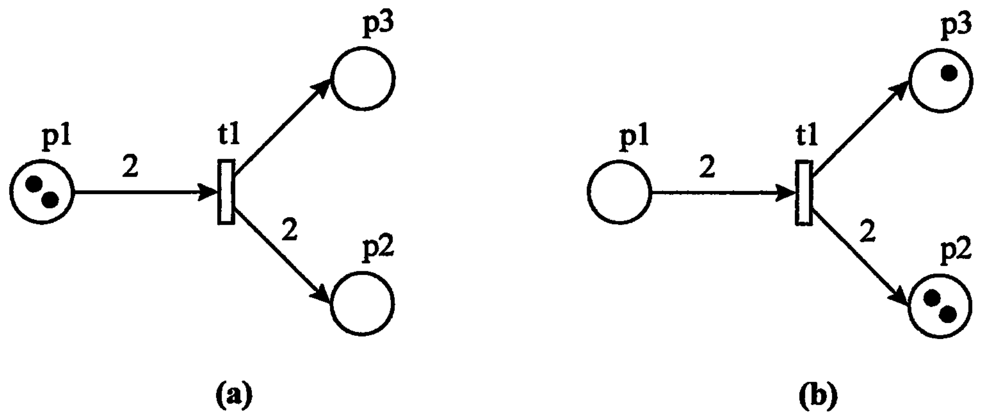


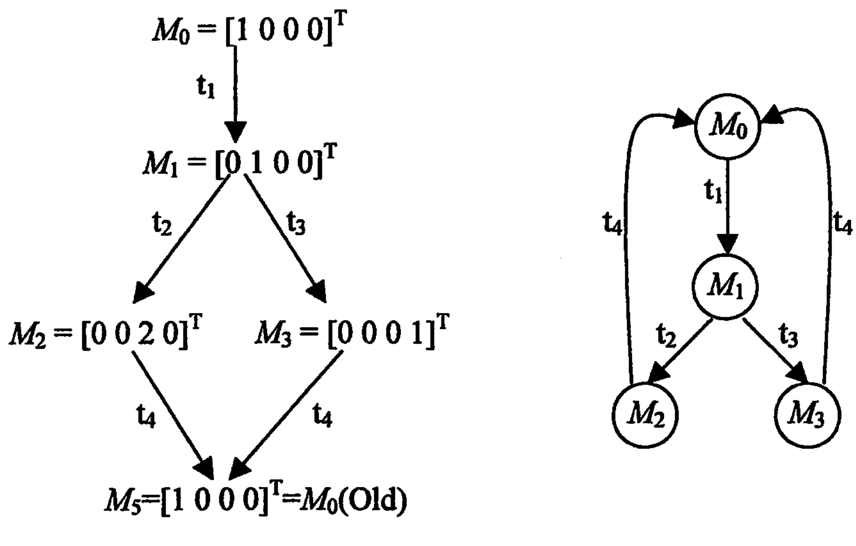
Figure 3.2 (a) Transition t_1 enabled. (b) Enabled transition t_1 fires

3.2.2 Analysis Methods of Petri Nets

The behavioral properties are useful for analyzing the modeled system. Methods of analysis for Petri nets may be classified into the following three groups: the coverability tree method, the matrix-equation approach, and reduction or decomposition techniques (Murata, 1989).

Coverability tree: From the initial marking M_0 of a Petri net, new markings can be obtained as many as the number of enabled transitions. This process results in a tree representation of the markings (reachability tree). Nodes represent markings generated from M_0 (the root) and its successors, and each arc represents a transition firing, which transforms one marking to another (Murata,1989). The goal of the reachability tree is to find all the markings which can be reached starting from M_0 (Proth & Xie, 1996, p.15). The coverability tree is derived from the reachability tree. At the expense of the loss of some information, the size of the coverability tree remains finite even if the size of reachability tree is not finite. Both trees are tools used to analyze the behavior of a Petri net.

Figure 3.3, shows the reachability tree and the coverability graph of Petri net, given in figure 3.1.



(a) **(b)**

Figure 3.3 (a) Reachability tree. (b) Coverability graph

Incidence matrix and state equation: The incidence matrix of a pure Petri net is an integer *nxm* matrix A, where n is the number of transitions, and m is the number of places (Zurawski & Zhou, 1994). A includes a_{ij} which is equal to $a_{ij}^{+} - a_{ij}^{-}$. a_{ij}^{+} is equal to number of arcs connecting transition t_i to its output place p_j, and a_{ij}^{-} is equal to number of arcs connecting transition t_i to its input place p_j. a_{ij}^{+} represents the number of tokens replaced on its output place, a_{ij}^{-} represents the number of tokens removed on its input place, and a_{ij} represents the change number of tokens in place p_j. Enabling rule can be formulated using this information. If $a_{ij}^{-} \leq M(p_j)$, i=1,2,...,m transition t_i is said to be enabled. Incidence matrix of Petri net model is;

$$A = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & -1 & 0 & 2 \\ 1 & 0 & -2 & -2 \end{bmatrix}$$

Incidence matrix shows the all connections in the net between transitions and places with weight of arcs. For example connection between p_1 and t_1 is represented in $A(1,1)$. –1 value explains that p_1 is input place of t_1 (because value is negative), and when the firing of t_1 one token leaves from p_1. Again, arc between t_3 and p_4 is shown in $A(3,4)$. 2 value represents the p_4 is output place of t_3 (because value is positive), and also 2 tokens are placed in p_4 when the t_3 fires.

The fundamental to this approach is the incidence matrix which defines all possible interconnections between places and transitions in a Petri net. The matrix equation is useful in studying the reachability problem. Any token condition in Petri net can be found easily using incidence matrix. This equation can be written as following;

$$M_k = M_{k-1} + A_i^T \quad k=1,2,\ldots$$

M_k is an mx1 column vector, and represents the k_{th} marking of Petri net. M_{k-1} is the previous token condition of M_k. t_i is the firable transition according to the M_{k-1}, and i

is the transition number. A_i is the i_{th} row of A, and is a 1xm row vector. If A_i^T is added to M_{k-1} then the new token condition M_k will be found easily. For model, given in 3.1, M_1= $[0\ 1\ 0\ 0]^T$, if M_3 is to be find, first of all, transition which changes from M_1 to M_3, must be determined. This movement is provided when t_3 fires. New token condition (M_3) is calculated using M_1 and third row of incidence matrix (A_3). If A_3=$[0\ -1\ 0\ 1]^T$ then $M_3 = M_1 + A_3^T = [0\ 1\ 0\ 0]^T + [0\ -1\ 0\ 1]^T = [0\ 0\ 0\ 1]^T$.

Also there are another two concepts related incidence matrix; *P*-invariant and *T*-invariant. They are useful in studying properties of Petri nets. *T*-invariant is an integer solution x of equation $A^T x = 0$. The nonzero entries in a *T*-invariant represent the firing counts of the corresponding transitions which belong to a firing sequence transforming M_0 back to M_0. Although a *T*-invariant states the transitions comprising the firing sequence transforming M_0 into M_0, and the number of times these transitions appear in this sequence, it does not specify the order transitions firings (Zurawski & Zhou, 1994). Another concept, *P*-invariant is an integer solution y of equation $Ay = 0$. The nonzero entries in a *P*-invariant represent weights associated with the corresponding places so that the weighted sum of tokens on these places is constant for all markings reachable from an initial marking (Zurawski & Zhou, 1994). Also subsets of transitions and subsets of places in the net can be found using *T*-invariants and *P*-invariants. *T*-invariants give the subsets of transitions, and *P*-invariants give the subsets of places.

3.3 Petri Nets in Manufacturing Systems

Petri nets have been used extensively to model and analyze manufacturing systems since modeling manufacturing systems by Petri nets provide following advantages: (Desrochers & Al-Jaar, 1995, pp.55-56; Zhou & Venkatesh, 1999, pp.41-42).

- Complex manufacturing system is modeled easily, because Petri nets capture the precedence relations and structural interactions of stochastic, concurrent, and asynchronous events, mutual exclusion, and precedence relations. In addition, their graphical nature helps to visualize such complex systems. They focus on local information rather global one by using bottom-up, top-down, and hybrid methods.

- Conflicts and buffer sizes can be modeled easily and efficiently. Also capacity overflow condition can be detected by not making any simulation study.
- Deadlocks in the system can be detected. This study does not require any simulation.
- Petri net models represent a hierarchical modeling tool with a well-developed mathematical and practical foundation.
- Various extensions of Petri nets, such as timed Petri nets, stochastic (timed) Petri nets, colored Petri nets, and predicate/transitions Petri nets, allow for both qualitative and quantitative analyses of resource utilization, effect of failures, and throughput rate, to name a few.
- Petri net models give a structured framework for carrying out a systematic analysis of complex systems. Various software packages have been developed for this purpose.
- Performance analysis without simulation can be made for many systems. Production rate, cycle time, resource utilization, reliability, and performability can be evaluated. Bottleneck machines can be identified.
- Discrete event simulation that can be driven from the model.
- Usefulness for scheduling problems in manufacturing system, because system conditions can be easily represented in a Petri net model.
- Finally, Petri net models can also be used to implement real-time control systems for a flexible manufacturing system. Hence, they serve as a replacement for programmable logic controllers.

In a manufacturing environment, places, tokens and transitions are usually interpreted in the following way (Desrochers & Al-Jaar, 1995, pp.57-58):

- Places represent resources such as machines, automatically guided vehicles, computer code, or even parts in a buffer. The existence of one or more tokens in a place represents the availability of particular resource; while no tokens indicates that the resource is unavailable. Therefore, the purpose of places is to capture the decentralized nature of the system and distributed state of the information in a complex manufacturing.

- A transition firing represents an activity. This activity begins and ends with two consecutive events. The time between events may be zero, corresponding to an immediate transition. For example, a machine repair is an activity that begins with the event, initiate repair. It is followed some time later by the event, repair completed. On the other hand, the activity downloading a part program is carried out by two events separated by zero time (for most practical modeling purposes).
- Places and transitions together represent conditions and precedence relations in the system's operation. For example, a token in a place implies that the condition is true, and no token, that is false.

Interpretations of places and transitions in manufacturing systems can be summarized in table 3.2. (Zhou & Venkatesh, 1999, p.74).

Table 3.2 Two Interpretations of Places and Transitions

PN Elements	First Interpretation	Second Interpretation
Places	Resource status and operation	Resource status and conditions
Transitions	Start and/or end of operations, process, activities, and events	Operations, process, activities, and events
Directed arcs	Material, resource, information, and/or control flow direction	

Petri net properties and their meanings in manufacturing systems are given in table 3.3. (Zhou & Venkatesh, 1999, p.75).

Table 3.3 Petri Net Properties and Their Meanings

PN Properties	Meanings in the Modeled Manufacturing System
Reachability	A certain state can be reached from the initial condition
Boundedness	No capacity (of, e.g., buffer, storage area, and workstation) overflow
Safeness	Availability of a single resource; or no request to start ongoing process
Liveness	Freedom from deadlock and guarantee the possibility of a modeled event, operation, process or activity to be ongoing
Reversibility	Re-initialization and cyclic behavior
Repetitiveness	Existence of repetitive operations/activities/events for some marking

Petri nets provide the modeling and analyzing of conditions in manufacturing systems. For example, production volume can be represented with tokens in a place, which is end of the production line. Product information or facility (e.g. AGV) movement can be also traced with related token movement in the net. Some manufacturing conditions and their representing in Petri net model are given in table 3.4. (Zhou & Venkatesh, 1999, p.74).

Table 3.4. Modeling Convention with Petri Nets

Concepts in Manufacturing	Petri net modeling
Moving or production lot size	Weight of directed arcs modeling moving or production Kanban
Number of resources, e.g., AGVs, machines, workstations, and robots	The number of tokens in places modeling its availability
Work-in-process	The number of tokens in places modeling the buffers and operations of all machines
Production volume	The number of tokens in places modeling the counter for the number of firing of transitions modeling end of a product
The time of an operation, e.g., setup, processing, and loading	Time delays associated with place or transition modeling the operation
Conveyance or transportation time	Time delays associated with directed arc, place, or transition modeling the operation
System state	Petri net marking (plus timing information for timed Petri net)
Sequence, concurrency, conflict, resource-sharing, etc.	Available Petri net class such as event graph, state machine, timed Petri net, and etc.

A comprehensive survey including the driving force behind PNs, qualitative analysis on Petri net models, PN simulation tools and etc. can be found in Zhou & Venkatesh, 1999.

3.4 Literature Review

In this section, a literature review including recent Petri net (PN) studies on modeling, deadlock prevention, controlling, scheduling, and performance analysis are presented.

3.4.1 Modeling

Cecil *et al.* (1992) made a review on Petri net applications in manufacturing. Also they modeled a manufacturing cell, which consists of input and output conveyors, one robot, and two CNC machines.

Cao & Sanderson (1994) described a decomposition of high level task sequences using a Petri net representation which facilities the analysis of robustness properties for the resulting plans. An approach was proposed for decomposition of assembly plan, which retains two important properties, traceability and viability. They used an AND/OR Petri net to represent task level sequences and follow a top-down hierarchical decomposition procedure based on Petri net to develop a more comprehensive robotic task representation. A robotic assembly system was modeled, and then this net is enlarged to level 1 Petri net using first decomposition algorithm. Level 2 Petri net model is occurred using other two decomposition algorithms. All decomposition algorithms, which all decomposition procedures in the study were formalized in, were given in paper. Petri net models in all levels were analyzed for safeness, liveness, and boundedness. They simulated the assembly system to determine the sequencing of tasks for each level by assuming that transition is fired only one time or none.

Shiizuka & Suzuki (1994) showed modeling power of Petri nets by modeling AGV networks in flexible manufacturing systems. Using colored Petri nets, they modeled behavior of AGVs such as, go straight ahead, junction of two ways, branch off three ways, junction of three ways, and etc. They also modeled an AGV network using all these behaviors. Study didn't include any numerical solutions.

Kiritsis & Porchet (1996) dealt with process planning using Petri net as a computer-aided design (CAD) or computer-aided process planning (CAPP). Proposed approach consists of three parts. First and second parts are to build a fixed format machining table and a Petri net model of the proposed model based on the machining table data, respectively. Third is to find a reachability graph of the PN model. Finally reachability graph is reduced, and optimum solution is found by applying simple heuristic. They illustrated their approach on a part of processing in a machine center. They divided the machining table in three regions, rough milling, drilling, and finishing operation. Petri net models and reachability graph were occurred for each region's operation. A simulation study was done to detect the possible problems in process planning stage. Final process plan, which is optimum, was found using reachability graph and the simple heuristic.

Jeng (1997) improved a Petri net synthesis theory, which has been proposed for modeling shared-resource automated manufacturing systems. Each sub system is modeled as a state machine using this theory. Common transitions, common transitions net and restrictions in merging were discussed. He also presented the types of circular waits in error net. An algorithm was given to determine whether the net might enter circular waits. Especially when the initial marking is large, this algorithm is more efficient than state enumeration techniques such as the reachability tree method. He applied his theory on an automated garmanent system, which includes two robots, two output buffers, garmanent handling mechanism, and a press. He built submodels for each of gandler, robots, garmanent and press, and then merges these submodels to construct a system net using common transitions and common transitions net. He analyzed this merged net for circular waits.

Ramaswany *et al.* (1997) interested development of multiple-input multiple-output (MIMO) subnets. Hierarchical time extended Petri nets (H-EPNs) provide to convert such a MIMO subnets to single-input single-output (SISO) net, which can be easily integrated with in a top-down decomposition. They discussed Petri net applications in manufacturing, advantageous and disadvantageous of bottom-up and top-down approaches and other decomposition techniques in details. Activator arc, place extensions, transition extensions, arc extensions and token extensions were explained under H-EPN section. They presented MIMO nets and MIMO-SISO transformation by giving definitions such as well formed subnet, successor, well defined block, MIMO subnet, QR sets, and etc. MIMO net properties were verified. Activator arcs were demonstrated in an assembly process consists of three workstations, and two robots. Assembly system includes scheduling, priorities and dynamic failure stations. System was modeled using H-EPN modeling methodology. First, a high level model was built using a top-down approach. Later, the various machine operations were modeled using a bottom-up synthesis technique. They occurred subnets for individual machine operations, priority scheduling of jobs, dynamic failure recognition and rescheduling of jobs. The final H-EPN model was created by integrating the equivalent SISO net with top level decomposition of the system model. SPNP simulation package is used to verify the properties of assembly system such as boundedness, liveness, and reversibility.

Adamou *et al.* (1998) dealt with modeling and controlling of flexible assembly systems using object-oriented Petri nets (OPN). Flexible assembly system was organized into hierarchically levels, which are factory, workshops, assembly line, cells, workcenter, stations, and operations, for knowledge acquisition and intelligence distribution. OPN coordinator models were generated for each material flow handling level, which were factory, workshop, assembly line, cell, and workcenter. Also OPN dispatcher models were created for factory, workshop, assembly line, cell, workcenter, and station or operation level. An assembly line architecture where each assembly line contained assembly cells including assembly workcenters was used to illustrate their approach. All OPN coordinator and dispatcher models were generated using Design/CPN, a PN software tool, developed by Meta Software Comparison. The flexible assembly system within computer integrated manufacturing implementation was designed for controlling of subassemblies.

Zha *et al*. (1998) proposed a knowledge-based Petri net (KBPN) approach, which combines an extension of the classic PNs with expert system, for modeling, planning, simulation, analysis, and evaluation of flexible assembly systems. In assembly sequence representation and generation stage, generating KBPN from AND/OR and PN graph was given for a simple assembly plan which includes three parts and three assembly levels. In KBPN planning stage, they used a hybrid strategy, which has concurrent and asynchronous event dispatching and continuous transition scanning. In the simulation and performance analysis stage, KBPN and assembly processes were observed. They used a simple flexible assembly cell including a part conveyor, a product output conveyor belt, a buffer station, an assembly station, and a robot manipulator, to analyze the performance of KBPN-based flexible assembly planning system. Subassembly time and total assembly time were determined as performances measures. They also developed a KBPN based expert planning system (KAPS) in visual Prolog 4.0 for understanding all functionalities of the KBPN model. Structure of KAPS, knowledge base design, inference engine, knowledge acquisition, and case-based self learning for KAPS were explained. Finally a mechanical model product with four components was analyzed using KBPN and KAPS, and subassembly time and total assembly time for different assembly sequences and assembly strategies were tested.

Zussman & Zhou (1999) proposed a disassembly process planning algorithm based on disassembly Petri net (DPN), which was developed by Zussman, Reiter, & Scharke in 1995. DPNs are the first complete framework for disassembly planning, execution, and adaptation to achieve the best results. They developed a procedure to determine the optimal disassembly plan using DPN. DPN concept and their process planning methodology were demonstrated on an AT&T's telephone. DPN model of the telephone built using given product specifications, represented all feasible disassembly process. The best disassembly process plan for the telephone was determined using their procedure.

Zha (2000) proposed object-oriented knowledge based Petri nets (OOKPN), which can integrate a knowledge based expert system and fuzzy logic into ordinary place/transition Petri nets to design concurrent intelligent product assembly processes. He also represented two class of PNs; Knowledge Petri nets (KPN) and Fuzzy knowledge Petri nets (FKPN). KPN focus on intelligent integration of product design, assembly evaluation, redesign for assembly, assembly process planning, design of assembly system, and assembly simulation. A Petri net tool based on C/C++ language and CLIPS object-oriented language, was developed for geometric modeling and design, planning and simulation, and evaluation. KPN, FKPN, and OOKPN were used for designing and planning of a micro switch, and its optimal assembly sequence was determined using developed PN tool.

Moore *et al.* (2001) proposed a Petri net-based approach which mechanically generates optimal or near optimal disassembly process plans for product recycling and remanufacturing, which consists of products with complex AND/OR relationships. In this study, they detailed their previous study (Moore *et al.*, 1998). Their approach consists of three steps; building the disassembly precedence matrix (DPM), obtaining the disassembly Petri net (DPN) from DPM, and giving the optimal or near-optimal disassembly process plan (DPP) from DPN. To generate the DPM they presented an algorithm using physical precedence relationships between assembly parts. Another algorithm was developed to obtain DPN from DPM. This algorithm finds the incidence matrix of DPN. They also developed reduced reachability tree (RRT) algorithm to determine near-optimal DPPs using DPN. The RRT algorithm produces the sets of partial disassembly sequence by doing a limited depth-first search, and then selects the best (or lowest cost) sequence set. In this study, they gave cost functions as time-based penalties, such as tool change penalty, disassembly direction penalty, and a penalty for delaying removal of hazardous component. To illustrate their approach they used a sample product consisting of five parts and two joining elements. They proved that RRT algorithm reduces the search space and gives a solution within a reasonable time.

Teng and Zhang (1993), Zhou *et al.* (1993), Ramaswamy & Valavanis (1994), Sung *et al.* (1994), Zuberek & Kubiak (1994a), Zuberek (1995), Seifodini & Zang (1996), Proth *et al.* (1997), Tiwari *et al.* (1997), Yan *et al.* (1997), Song & Lee (1998), Wang & Wu (1998), Zimmerman & Hommel (1999), Zha & Lim (2000), Tang *et al.* (2001), & Zha *et al.* (2001) are also some of the worthy articles including modeling studies of manufacturing systems using PNs.

3.4.2 Deadlock Prevention

Liveness property is helpful to determine the deadlock points and to prevent the system from deadlocks in designing stage. Some of the recent studies on deadlock prevention are given in the following paragraphs.

Ezpeleta *et al.* (1995) focused on deadlock prevention/avoidance problem. They adopted Petri net as a tool for modeling the dynamic systems. They defined the class of the Simple Sequential Processes (S2P); then extended it to model the use of resources (the class of S2PR) and, finally defined the class of Simple Sequential Process with Resources (S3PR). Liveness is analyzed for S3PR models, and a control policy that uses siphons is defined for deadlock prevention. A production cell, which consists of three robots (each one can hold a product at a time), four machines (each one can process two product at a time), three loading buffers, and three unloading buffers, is modeled as a S3PR model. Then, proposed deadlock control policy is applied on S3PR model to check its liveness property.

Chu & Xie (1997) proposed to exploit the potential of siphons and traps for deadlock related analysis of Petri nets. A Petri net without potential deadlock is deadlock free. They used the mathematical programming approach for examination of all minimal siphons that become empty. They presented some characteristics of minimal siphon to narrow the solution space. They gave two examples; one of them was automated manufacturing system which includes four machines, two robots, two buffers of capacity three and an assembly cell, another was a job-shop system which consists of a set of resources and a set of part types. Mixed-integer programming (MIP) was used to detect minimal siphons, thus potential deadlocks were determined. Also they searched effects of their suggestion on asymmetric choice nets and augmented marked graph. They proofed that an asymmetric choice net is live if and only if it is potential deadlock free, and an augmented marked graph is live and reversible if and only if it is potential deadlock free.

D'Souza & Khator (1997) interested in deadlocks in automated manufacturing systems. Deadlocks are detected and avoided by performing an off-line analysis and re-allocating buffer capacity at the critical workstations. They searched deadlocks in a computer integrated assembly system (CIAC), which consists of three workstations, AS/RS and transfer table for assembling of a single job. Each workstation has one input and one output buffer, and capacity of each buffer is one assembly. Assembly parts are loaded from AS/RS to CIAC using the transfer table. A PLC was used for control of CIAC. CIAC was modeled using marked graph, and then model was transformed into Petri net in Matrix Definitional Form (MDF). MDF is necessary for application of Deadlock Detection Algorithm (DDA). They determined the deadlocks in the cell using DDA. For solving deadlock problem buffer capacity of critical workstations is increased using Buffer Re-allocation Algorithm (BRA). After marked graph was transformed into GSPN, GreatSPN, a software package, was used to test that if the system performance decreases as buffer capacity increases. They observed changes in throughput rate for different input rates and different batch size of assemblies. They determined the effective batch size, which provides the effective throughput rate.

Ezpeleta & Colom (1997) dealt with modeling large systems and control policies for deadlock prevention. They modeled the automated manufacturing cell, which was given in study of Ezpeleta *et al.* (1995), using their proposed modeling methodology. system architecture (facility layout) and process plans (alternative part routings) were separately modeled then both models were integrated using colored Petri nets. For final model a deadlock prevention policy, developed in the study of Ezpeleta *et al.* (1995), was used.

Wang & Wu (1998) presented a new paradigm, called colored timed object-oriented Petri net (CTOPN), which is used to model an automated manufacturing system (AMS) consists of one lathe, one milling machine, one input/output buffer, one work in process (WIP) buffer linked by a robot, and processing of two part types with different routings. CTOPN model was derived from the object-oriented Petri net (OPN) model of AMS. A deadlock detection algorithm to detect deadlocks in CTOPN model is used. This paper also comprised updating CTOPN model by simply adding new place and transitions which describe the new conditions in the system, a sequencing study minimizing the makespan by a search algorithm, transforming the OPN model into control model by adding control logics to OPN model.

When the routing is flexible, the problem of deadlock avoidance in FMSs becomes an open problem. Wu (1999) developed a Petri net to describe the problem, called as colored resource oriented Petri net (CROPN). Essential characteristic of the token in a place is defined to avoid complexity. For deadlock avoidance in FMSs, the necessary and sufficient conditions and corresponding control policy based on CORPN was presented in this study. The control law imposes minimal restriction on the scheduler. This is a dynamic policy of resource allocation and can be implemented in real-time. Therefore there can be as many parts as possible in the system while deadlock is totally avoided.

Tiwari *et al.* (2001) were interested in deadlocks and conflicts in an AGV system which has two AGVs responsible for transporting two products among three processing stations in a FMS. System was modeled using timed Petri net and activity cycle diagram to model the dynamic characteristic of FMS, and the movement of

AGVs and part types, respectively. Behavioral properties of Petri net model such as liveness, deadlock free, and boundedness were checked. The system's performance was analyzed for some performance measures such as mean utilization and utilization index. They compared their results with some existing studies in the literature.

3.4.3 Controlling

Real-time controlling needs determining the system conditions, immediately so that the system could be brought into normal conditions. Petri nets were widely used for controlling. Some of them are given in this section.

Venkatesh & Ilyas (1993) interested with controlling of FMS network. Manufacturing systems in this study consists of four pneumatic pistons. They built two Petri net models using augmented timed Petri nets. They also developed a software package which simulates the Petri net models and generates certain performance criteria of models.

Lee & Jung (1994) presented a new methodology for flexible operation planning using Petri net approach. They modeled a hole making process using Petri net, and then determined the operation selection and sequencing from the PN model. A subset of the PN model gave operation selection and sequencing of the hole making process.

McCarragher B. J. (1994) presented a new approach to process modeling, task synthesis, motion control and trajectory planning for robotic assembly which was modeled as discrete event dynamic system using Petri nets. His discrete event control approach could solve monitoring, synthesis, task-level control, and trajectory planning problems. He used this approach on a control system of a robotic assembly consists of process monitor, discrete even controller and the trajectory planner. This approach to robotic assembly proved to be highly successful in detecting and recovering situations from undesirable contact states resulting from misalignment or mismatch between the model and the actual system.

Venkatesh *et al.* (1994) proposed a new class of Petri nets, real-time Petri nets. They made a comparison between ladder logic diagrams applied in various industries for real-time controlling, and real time Petri nets to control an industrial automated system, which is the same system in Venkatesh & Ilyas (1993). Real-time Petri nets were used for sequence of controllers. Comparison results were given according to the sequence number.

Yasuda & Tachibana (1994) modeled a multiple robot system for controlling. Robotic tasks represented a sequence of primitive actions in modeling stage. Study presented redesign and implementation of a parallel distributed control architecture for industrial multiple robot system using Petri nets.

Thomas *et al.* (1996) presented a type of Petri net to represent different levels in the assembly hieararchy, such as assembly plan and control plan level.

Mo & Tang (1998) designed and analyzed a task oriented robot task control system based on manufacturing message specification (MMS), which is used for shop floor machine control, using Petri nets. The top level Petri net was derived from the MMS process and subsequently was decomposed by the top down approaches into detailed subnets. Petri net model included functions of client and server stations.

Moore & Gupta (1999) developed a stochastic colored Petri net (SCPN) model for traditional kanban control policy, and flexible kanban control policy. When traditional policy is employed, problem is to find optimum number of kanbans when the number of kanbans is fixed throughout the production cycle whereas when flexible policy is employed problem is to find when and how much the number of kanbans is to be changed to improve the system performance. SCPN models, which were live and bounded, were obtained for two card kanban systems. Moore & Gupta represented formulas to determine work in process for each kanban control policy by using boundedness property. These processes were made for both the kanban control policies. SCPN models were simulated using a SCPN tool for different 15 cases, under three scenarios and same assumptions. Systems' performances were analyzed, and performances of control policies were compared.

Yeung & Moore (2000) proposed a methodology integrating Genetic Algorithms with Colored Petri nets (CPN) to control faults in flexible assembly systems. Genetic Algorithm was used to sequence job orders, to select working operation plans into flexible process plannings, and to assign operations to workplaces and machines. Design of fault tolerant cell control for flexible assembly systems was done using their proposed algorithm which is CPN-based cell control. They tested their methodology on a flexible assembly system with a conveyor.

Ezpeleta *et al.* (1995), Seifodini & Zhang (1996), D'Souza & Khator (1997), Ezpeleta & Colom (1997), Yan *et al.* (1997), Adamou e al. (1998), Wang & Wu (1998), and Wu (1999) can be considered as studies in which Petri nets are used for control.

3.4.4 Scheduling

Many industrial systems include several resources and jobs sharing these resources. Effective scheduling of these systems provides more throughput rate, in other words less idle time. Researchers have used reachability property to solve scheduling problem. Some scheduling algorithms are adopted in Petri nets, and also some heuristic approaches are developed. Some of the recent studies in this area are given below.

Lee & DiCesare (1994a) presented a new scheduling method by combining Petri net model and heuristic search. This method gives optimal or near optimal scheduling results. They tested this method on an FMS having three machine, one robot, and four jobs. They generated Timed Place Petri net models for each job, and then tested the models for deadlocks. They used L1 algorithm, which was adopted from the well known graph search algorithm A*, to solve scheduling problem of FMS.

Lee & DiCesare (1994b) presented two AGV models, which integrated the control of part processing facilities such as machines and robots. Two AGV system models cover both cases where an AGV is assigned to a job until the job is completed (centralized AGV systems), and a case where each machine has its own AGV (distributed AGV systems). Manufacturing area includes five AGVs, four jobs and three machines. They built Petri net models for centralized and distributed AGV systems. To solve the scheduling problem of both AGV systems, they used a scheduling algorithm called L1, and evaluated its performance for makespan. They also showed that L1's performance, for makespan, was better than the performances of three heuristic dispatching rules, which are shortest queue length, shortest process time, and shortest waiting time.

Sun *et al.* (1994) proposed two basic submodels, a transportation model and a process flow model, to construct the timed place Petri net model of an FMS consists of load/unload station, AS/RS, AGVs, robot, lathe, milling machine, a machine center, and three job types. Transportation model describes the behavior of AGV while other model describes the behavior of part routing and resource assignment. Transportation models of transportation layout module, movement control module and push control module were built for this FMS. They used a push AGV strategy to solve the vehicle collision and traffic jam problems in the system. To solve the nondelay scheduling problem of this FMS they used limited expansion A algorithm, minimizing makespan, which is then modified general A* algorithm (L1). Results of makespan were given for different job sizes.

Zuberek & Kubiak have many articles on scheduling with Petri nets. In one of their papers, Zuberek & Kubiak (1994a), they used timed Petri net to model a flexible manufacturing system, which includes three machines, one robot, and two conveyors; for input parts and output parts, respectively. All possible schedules were explained and two schedules of them were modeled using Petri net. *P*-invariants, which give the subnets of the whole Petri net, were found for these two PN models, and then cycle times of subnets were determined. Schedules are made according to the subnet, which is given optimal cycle time.

In colored Petri net models of manufacturing cells, colors can be used to represent different schedules at the same cell. Therefore, analysis of several schedules can be performed simultaneously. Zuberek (1995) used conflict free colored nets to model the manufacturing cell, which was given in Zuberek & Kubiak (1994a). For this model, he determined six schedules so that each one has one color. There were totally thirty-three *P*-invariants for six schedules. The minimum cycle time of each schedule is found by the *P*-invariants of subnets.

Zuberek (1996) presented a systematic method of generating all composite schedules. In a composite schedule, several (identical or different) parts enter and then leave a manufacturing cell in each cycle. Each schedule is modeled using timed Petri net, and its performance is determined by invariant analysis of the derived Petri net model. He illustrated this method on the same example in his previous study (Zuberek & Kubiak 1994a). He generated composite schedules for different initial configurations, decomposed some of the schedules into a pair of simple schedules, computed the cycle time of each schedule, and found optimum schedule. He translated these schedules into sequences of robot actions.

Tiwari *et al.* (1997) proposed a heuristic for machine loading problem by reducing the system imbalance and tail after by maximizing throughput rate. Their proposed heuristic uses shortest processing time (SPT) rule for scheduling, and requires some assumptions such as; non splitted jobs, unique job routings, non shared and duplicated tool slots, parts, pallets and fixtures always available to reduce complexities in the problem. They solved loading problem of a sample system, which consists of four machines, five slots, and eight jobs with different numbers of sub jobs and with different batch sizes using their heuristic approach. They modeled the system using Petri nets to verify the proposed heuristic approach and to observe dynamic behavior of this approach, and compared their proposed approach with two heuristics, which are developed by Shankar & Srinivasulu in 1989 and Mukhapadyay *et al.* in 1992, for ten problems in the literature.

Yan *et al.* (1997) proposed a new class of Petri nets, extended high level evaluation Petri net (EHLEP-N), which is a combination of expert systems and Petri nets, for real time scheduling and control of FMSs. The use of EHLEP-N was illustrated on an FMS, which consists of one NC machine, three machining centers, one vehicle, four capacitated buffers, and one capacitated work piece storage. This FMS was analyzed for two production plan, one includes deterministic processing times other includes normally distributed processing times. Both plans include two policies, rescheduling and non-rescheduling. Yan *et al.* computed total processing times, makespans, and utilization rates of each machine for two policies, and showed that rescheduling policy increased the throughput rates in most cases.

Song & Lee (1998) investigated the scheduling problem for cyclic job shops with blocking. They used marked graphs to develop Petri net models for cyclic job shops with alternative buffer strategies. Petri net models were analyzed for deadlock detection, and cycle times of PNs were computed. They presented a mixed integer model to find an optimal deadlock free schedule which minimizes the cycle time of PNs. A machine cell, consists of six machines and six jobs, is used for testing scheduling results by chancing buffer capacities.

Zuberek (1998) described an approach for systematic derivation of schedules for manufacturing cells. His approach is based on refinement net that provides enlargement of a Petri net model from simple to complex. According to this approach, first model is built for empty cell, and then model is enlarged for one machine in the cell using refinement net, and so on. He applied his approach on a manufacturing cell, which is given in Zuberek & Kubiak (1994a). In this study he assumed that all parts follow the same path through the cell, including three machine, two conveyor, and constructed Petri net model of the manufacturing cell for scheduling using timed Petri nets and refinement nets. Refinement nets provided enlargement of Petri net model from one machine to three machines. Performance evaluation of generated schedules was not included in this paper.

Jeng *et al.* (1999) proposed a heuristic search method based on Petri nets for scheduling of flexible manufacturing systems with assembly (FMSA) by using reachability analysis. The proposed algorithm was derived from the traditional search algorithm A*, and its goal was to find the minimum makespan. The algorithm searches the routes into reachability tree, and then selects the best route, which is given or is expected to give minimum makespan. Reachability tree is generated from generalized symmetric nets (GSN) and generalized asymmetric nets (GAN) models of FMSA. GSN and GAN developed by Jeng and Lin in 1997 are two classes of timed place Petri nets to model FMSA. The proposed heuristic was based on GAN structures and dynamics. They made a simulation study to compare their heuristic with two heuristics presented in Jeng & Chen (1998) and Lee & DiCesare (1994a). Simulation was performed on three examples. In the first example, 1000 of GSNs and GANs were randomly generated, where no assembly tasks were modeled. Second example was derived from first example by adding assembly. Third example was obtained by setting dynamic information into second example. More test cases were randomly generated for all examples. According to the simulation results for three examples, the proposed heuristic was better than two methods in about 60-95% of the test cases

Yee & Ventura (1999) used assembly-marked Petri nets to find optimal assembly sequences with some assembly operation constraints such as ease of component handling, ease of component joining, and tool changes. Ease of component handling in a sequence is related with physical characteristics of the components such as shape, size, volume, and weight. Ease of component joining determines the direction changes in assembly operations. Required tool operations in assembly are explained in tool changes. All constraints were modeled in an integer linear programming formulation, where the goal was to minimize assembly time. To solve the model, they used an exist subgradient optimization algorithm and dynamic programming algorithm based on assembly Petri net developed by Yee & Ventura in 1999. Their solution procedure was applied to three types of products, which were testing device, switch, and pressure recorder. Solutions were compared with LINDO's results. Sequence results were the same but their procedure was faster then LINDO.

Frey (2000) introduced a two-step assembly line sequencing method based on Petri nets. In the first step, an assembly plant is modeled as a discrete event system, and then a branch and bound algorithm is used to obtain Petri net model of the system. Petri net model is reduced by aggregating the states and events, which were represented in places and transitions, respectively, and by grouping the events. In the second step, a feasible sequence for job set is obtained. Frey demonstrated this method on an example, tested it on a real data from a German car manufacturing system, and showed that it is less complex and more speed than mixed integer linear programming.

Zha & Lim (2000) developed Expert Petri net (EPN), integrating the Artificial Intelligence (AI) concepts into ordinary Petri nets. EPN allows using knowledge annotations in ordinary Petri nets by adding control places, time constraints, and place and transition knowledge annotations. The methods for knowledge representation in AI such as the predicate logic, if-then production rule, frame, and object orientation can be used to annotate the knowledge in EPN. Descriptive and fact knowledge, rule knowledge, and control knowledge were represented in places and transitions. EPN also provides the knowledge-based event-driven search and control technique by working in a simple hybrid inference which consists of forward inference (breadth-first search) and backward inference (depth-first search). New Petri net classes can be easily derived from EPN, such as colored EPN. They presented an algorithm to generate EPN from ordinary PN. They used EPN for assembly/disassembly task analysis, modeling and plan representation. Task planning problem can be defined as a reachability problem, and then sequence of the task can be obtained using EPN by forward and backward searching. A high-level EPN, which includes complete representation of system states, gives the optimal assembly/disassembly sequence with minimum time or cost. Low-level EPN is decomposed from high-level EPN for subsequencing of assembly operations using their proposed decomposition algorithm. EPN implementation for assembly/disassembly system was illustrated on a robotic optic lens assembly. High-level and low-level EPN models were generated, and then a simulation study was made to observe total assembly time under the same sequence and different sequences. They also proposed a prototype system, which is a knowledge based intelligent system, for EPN based assembly/disassembly planning. Expert Petri net based assembly/disassembly planning system (EPN-ADPS), which was implemented in Visual Prolog, automatically generates, selects and evaluates, and optimizes the assembly or disassembly plans.

Tang *et al.* (2001) developed an integrated approach for disassembly planning and demanufacturing operation in an integrated flexible demanufacturing system using Petri nets. They proposed three Petri net models; Product Petri Net (PPN), which shows all feasible disassembly sequences, subassemblies, and components and the end of life value of products; Workstation Petri Net (WPN), which models the condition of workstations and provides the availability information of workstations; and

Scheduling Petri Net (SPN), which is used for machine scheduling. They developed an algorithm to determine the optimal disassembly path (disassembly planning). This algorithm uses information from WPN (for workstation condition) and PPN (for product structure and product information). They proposed another algorithm for scheduling of assembly processes in the workstations, and applied it on disassembly of a PC.

More scheduling studies could be found in articles of Cao & Sanderson (1994), Wang & Wu (1998), Zha *et al.* (1998), Yeung & Moore (2000), and Zha (2000).

3.4.5 Performance Analysis

Petri nets provide some performance analysis such as cycle time, throughput rate, and utilization rate. For example event graphs are very useful class of Petri nets for determining the cycle time. In the following, recent papers including performance analysis studies are given.

Teng & Zhang (1993) developed a procedure based on the net's incidence matrix for decomposition of Petri net models. They explained the decomposition procedure in details, and used different examples to apply the procedure. First example was a flow line manufacturing system consists of three machines and two capacitated buffers. They modeled this system such a marked graph. Second example was a flexible manufacturing cell, which consists of four workstations. This system was modeled as a state machine. Last example was an assembly manufacturing system. They built the subnets of each model using proposed decomposition procedure. They discussed in details the differences between the whole nets and their subnets from the point of view of simulation time, place and transition utilizations, and throughput rates.

Zhou *et al.* (1993) applied Petri nets on a real-life FMS cell. This cell consists of a handling conveyor system that includes four carts, CNC milling machine, robot used for loading and unloading operations, part presentation, computer vision system, a drilling workstation that has a robot, and PLC that coordinates all the movements of the individual components. For modeling of the cell they used system decomposition technique such a Petri net design method. They decomposed the system into the following parts; 1- conveyor system, 2- CNC milling machine, robot, computer vision system and parts presentation station, and 3- drilling workstation. They modeled each part using Petri nets, and found qualitative properties such as liveness, safeness, and reversibility. Final Petri net was obtained by integrating these submodels. Integrated model was converted to event graph model which is live, safe and reversible. Event graph model was reduced to decrease the number of circuits in the system, and then cycle time of FMS cell was calculated using reduced model.

Barkoui *et al.* (1994) used high level Petri nets for tool management in an FMS. Tool management (or selection) is a resource sharing problem. They modeled the tool selection procedures using stochastic Petri nets under the system's dynamic behaviors. In performance analysis stage, a ratio, which is tool preparation time/batch processing time, was tested to observe variation of production rate for different number of machines (2,4,8) and for sharing and nonsharing polices.

Ramaswany & Valavanis (1994) used extended Petri nets (EPN) to model, analyze, accommodate and simulate potential failures in a material handling system (MHS). Procedure of the analysis and accommodation of failures consists of three steps; 1) failure is identified as soon as possible when it occurs, 2) failure is accommodated by rectifying at the lowest level nets, 3) if step 2 is not possible, failure is accommodated in the next high level net. EPN model provides for running this procedure correctly. MHS consists of AS/RS, conveyors, stations, robots, and vision system (cameras and laser spreader). System operates on two different modes; mode 1 is the loading of the AS/RS bays, mode 2 is the unloading of objects from bays to pallets. They modeled the system without failures as EPN model. Potential failures in the system were explained in details. The failures were added to model by using a procedure with five steps. In the first step, action charts of all operations in the system are constructed. The charts provide for identifying the potential failure areas in the system. Second step includes generating action with failure detection and recovery, and third step includes identifying various subsystem areas and common operations, and building related subnets at lowest level. Following step includes identifying failure on different levels, and finally each net is modeled and analyzed independently. Using this procedure, they built subnets for requesting and releasing of camera, moving objects to destination area, picking up object, actions on the input conveyor, moving identified objects to AS/RS area, moving undefined objects to reject pallet area, accommodating pick up failures in mode 1, reidentifying objects in reject pallet area, operation in mode 2. All subnets which were safe, live, and reversible, and the nets related to each of operational mode and the final whole system were simulated individually using SPNP software.

Zuberek & Kubiak (1994b) dealt with throughput analysis of nets. They modeled a system given in their previous study (Zuberek & Kubiak, 1994a) using bounded conflict free nets. They used scheduling results obtained by the previous study, modeled each sequence, and then reduced these models using net transformations. They computed the throughputs of two sequences. Zuberek & Kubiak (1999) confirmed this study and previous study.

Seifodini & Zhang (1996) paid attention to real-time control of manufacturing systems. They proposed a new approach based on the analysis of the system structure and parallel properties. The main components of this new approach are Petri net modeling and decomposition, and parallel discrete simulation. They applied their approach on a manufacturing system with three manufacturing cells. Each cell has one or more milling machines, one deburring workstation and one robot. There are two buffers between cells. Three different types of parts are processed. Each part has a proportion of production. System was modeled under these constraints using Petri nets. Parallel relations of the subnets obtained using decomposition procedure were analyzed, and these subnets were simulated under determined proportion values. Their approach reduced simulation time. They estimated buffer utilizations using the simulation results.

Alpan & Jafari (1997) dealt with dynamic analysis of timed Petri nets, and aimed to solve a resource-sharing problem, which includes two processes sharing a resource. They explained the conflicts of the system; both of the processes need to resource at the same time, and waiting times; a process waits for completing of other process, and then cycle times of processes were found. They presented a new relative temporal analysis assuming no conflicts. A control chart mechanism was developed to eliminate avoidable waiting. For two case studies, they made sequences with no conflicts, using an algorithm, which is based on results of the relative temporal analysis and finds the natural sequence of the system. They were occurred sequences for different processing times.

Determining the minimal numbers of AGV is one of the important steps in FMS design. Proth *et al.* (1997) proposed a branch and bound (B&B) approach based on event graphs to define the minimal numbers of AGV when the productivity of the system is maximized. In this study, productivity of the system was expressed in terms of the cycle time. Smaller cycle time provides greater productivity. They modeled the problem as a linear programming model, and then an alternative formulation was developed using linear programming model. New formulation was used for B&B algorithm. They applied the B&B algorithm to the design of a FMS, which consists of four machines, producing three product types with different manufacturing process and different production ratio. The goal was to reach given productivity using minimum number of AGV. FMS was modeled as an event graph. Circuits in the event graphs were classified as process circuits, command circuits, and mixed circuits. They dealt with tokens in process circuits, because these tokens represent AGVs in the FMS. Using their approach, they found an initial marking which allows reaching the minimal cycle time.

Zimmermann & Hommel (1999) proposed a new modeling technique for manufacturing systems based on colored Petri nets. A Petri net model of manufacturing system usually includes both the structural information of the system and the specification of the production routes. Such an integrated model must be redefined if the production route of a single work piece changes. Their technique provides to model separately the production routes and the structure of manufacturing system. Therefore manufacturing system can be analyzed for more than one work piece. They applied their technique on an automated manufacturing system, which includes an assembly station. Machine failures were added to the model. This system was modeled and analyzed using TimeNET software, the changes in throughput were searched using alternative numbers of carriers, and machine utilizations and the impact of the machine failures on overall performance were analyzed. According to the results, they suggested new policies to provide implementation on throughput. These policies made an increase in throughput by 22 %.

Zha *et al.* (2001) developed a concurrent intelligent design approach and framework, based on previous studies (Zha *et al.*, 1998, and Zha, 2000), for robotic assembly systems using the Knowledge Petri Net (KPN). KPN is employed for mechanical systems and assemblies modeling and design, and process planning and design. Zha *et al.* organized flexible assembly systems (FAS) modeling with KPN into four steps. In the first step, hierarchical level abstractions for FAS are determined. KPN model for FAS is generated in the second step. Third step includes validation studies of KPN. Behavioral analyses of KPN for FAS are done in the last step. They also obtained a function-behavior-structure model, which have been widely used for design of mechanical systems and assemblies in the literature, using KPN. Functional model, structure model, behavior model and performance of assembly systems were explained. They also represented a design implementation for robotic assembly system. The implementation consists of functional analysis and decomposition, design prototypes and generation of alternative system structures. A micro optic lens product assembly system was modeled and simulated using KPN and POSES++, respectively. Performance measures in the study were cycle time, capacity, availability, and productivity. For this assembly system, various simulation runs were performed for four different system structures. The assembly system was evaluated and its layout was analyzed according to the simulation results.

Zuberek (1996), D'Souza & Khator (1997), Moore & Gupta (1998), Zha *et al.* (1998), and Tiwari *et al.* (2001) are some of the other researches on performance analysis of manufacturing systems.

Recent studies on Petri net applications in manufacturing systems were given previous subsections. All studies are summarized in table 3.5 according to the year of the study, researcher(s), aim(s) of the study, manufacturing system in the study, Petri net class used in the study.

Table 3.5 Petri Net Applications in Manufacturing Systems

Year	*Researcher(s)*	Aim(s) of Study	*Manufacturing System*	Petri Net Class
1992	*Cecil et al.*	Modeling	Flexible Manufacturing Cell	Petri Nets
1993	*Teng & Zhang*	Modeling, Performance Analysis	Flow Line, Flexible Manufacturing Cell, Assembly Manufacturing	Marked Graph, State Machine
1993	*Venkatesh & Ilyas*	Controlling	Flexible Manufacturing System	Augmented Timed Petri Nets
1993	*Zhou et al.*	Modeling, Performance Analysis	Flexible Manufacturing Cell	Marked Graph, Petri Nets
1994	*Barkoui et al.*	Performance Analysis	Flexible Manufacturing System	Stochastic Petri Nets
1994	*Cao & Sanderson*	Modeling, Scheduling	Robotic Assembly System	AND/OR Petri Nets
1994a	*Lee & DiCesare*	Scheduling	Flexible Manufacturing System	Timed Place Petri Nets
1994b	*Lee & DiCesare*	Scheduling	Flexible Manufacturing System (with AGV)	Timed Place Petri Nets
1994	*Lee & Yang*	Controlling	Computer Integrated Manufacturing	Petri Nets
1994	*McCarragher*	Controlling	Robotic Assembly System	Petri Nets
1994	*Ramaswany & Valavanis*	Modeling, Performance Analysis	Material Handling System	Extended Petri Nets
1994	*Shiizuka & Suzuki*	Modeling	Flexible Manufacturing System	Colored Petri Nets
1994	*Sun et al.*	Modeling, Scheduling	Flexible Manufacturing System	Timed Place Petri Nets
1994	*Venkatesh et al.*	Controlling	Automated Manufacturing System	Real-Time Petri Nets
1994	*Yasuda & Tachibana*	Controlling	Multiple Robot System	Petri Net
1994	*Zuberek & Kubiak*	Modeling, Scheduling	Manufacturing Cell	Timed Petri Net, Conflict Free Nets
1994	*Zuberek & Kubiak*	Performance Analysis	Manufacturing Cell	Timed Petri Net
1995	*Ezpeleta et al.*	Deadlock Prevention, Controlling	Flexible Manufacturing Systems	Simple Sequential Process with Resources Petri Nets
1995	*Zuberek*	Modeling, Scheduling	Manufacturing Cell	Conflict Free Colored Petri Nets
1996	*Kiritsis & Porchet*	Modeling	Flexible Manufacturing System	Petri Nets
1996	*Seifodini & Zhang*	Modeling, Controlling, Performance Analysis	Manufacturing Cell	Petri Nets
1996	*Thomas et al.*	Controlling	Automated Assembly Systems	Assembly Petri Nets
1996	*Zuberek*	Scheduling, Performance Analysis	Manufacturing Cell	Timed Petri Nets
1997	*Alpan & Jafari*	Performance Analysis	Resource Sharing	Timed Petri Nets

Table 3.5 (Cont.)

Year	*Researcher(s)*	**Aim(s) of Study**	***Manufacturing System***	**Petri Net Class**
1997	*Chu & Xie*	Deadlock Prevention	Automated Manufacturing Systems, Job-Shop	Asymmetric Choice Petri Nets, Augmented Marked Graph
1997	*D'Souza & Khator*	Deadlock Prevention, Controlling, Performance Analysis	Computer Integrated Assembly System	Marked Graph, Generalized Stochastic Petri Nets
1997	*Ezpeleta & Colom*	Deadlock Prevention, Controlling	Automated Manufacturing Cell	Colored Petri Nets
1997	*Jeng*	Modeling	Automated Manufacturing Systems	Petri Nets
1997	*Proth & Xie*	Modeling, Performance Analysis	Flexible Manufacturing System	Event Graph
1997	*Ramaswany et al.*	Modeling	Assembly Line	Hierarchical Time Extended Petri Nets
1997	*Tiwari et al.*	Modeling, Scheduling	Flexible Manufacturing System	Petri Nets
1997	*Yan et al.*	Modeling, Controlling, Scheduling	Flexible Manufacturing System	Extended High Level Petri Nets
1998	*Adamou et al.*	Modeling, Controlling	Flexible Assembly Systems	Object-Oriented Petri Nets
1998	*Mo & Tang*	Controlling	Flexible Robot System	Petri Nets
1998	*Moore et al.*	Modeling	Disassembly Systems	Disassembly Petri Nets
1998	*Song & Lee*	Modeling, Scheduling	Cyclic Job Shop	Petri Nets
1998	*Wang & Wu*	Modeling, Deadlock Prevention, Controlling, Scheduling	Automated Manufacturing System	Colored Timed Object-Oriented Petri Nets
1998	*Zha et al.*	Modeling, Scheduling, Performance Analysis	Flexible Assembly Manufacturing Cell	Knowledge-based Petri Nets
1998	*Zuberek*	Scheduling	Manufacturing Cell	Timed Petri Nets
1999	*Jeng et al.*	Scheduling	Flexible Manufacturing Systems with Assembly	Timed Place Petri Nets, Generalized Symmetric Nets, Generalized Asymmetric Nets
1999	*Moore & Gupta*	Controlling, Performance Analysis	Kanban	Stochastic Colored Petri Nets
1999	*Wu*	Deadlock Prevention, Controlling	Flexible Manufacturing Systems	Colored Resource Oriented Petri Nets
1999	*Yee & Ventura*	Scheduling	Automated Assembly Systems	Assembly-Marked Petri Nets
1999	*Zimmermann & Hommel*	Modeling, Performance Analysis	Automated Assembly System	Colored Stochastic Petri Nets

Table 3.5 (Cont.)

Year	*Researcher(s)*	Aim(s) of Study	*Manufacturing System*	Petri Net Class
1999	*Zussman & Zhou*	Modeling	Disassembly Processes	Disassembly Petri Nets
2000	*Frey*	Scheduling	Assembly Lines	State Machine
2000	*Yeung & Moore*	Controlling, Scheduling	Flexible Assembly Systems	Timed Colored Petri Nets
2000	*Zha*	Modeling, Scheduling	Assembly/Disassembly Systems	Knowledge Petri Nets, Fuzzy Knowledge Petri Nets, Object-Oriented Knowledge based Petri Nets
2000	*Zha & Lim*	Modeling, Scheduling	Assembly/Disassembly Systems	Expert Petri Nets
2001	*Moore et al.*	Modeling	Disassembly Systems	Disassembly Petri Nets
2001	*Tang et al.*	Modeling, Scheduling	Integrated Flexible Demanufacturing System	Product Petri Nets, Workstation Petri Nets, Scheduling Petri Nets
2001	*Tiwari et al.*	Deadlock Prevention, Performance Analysis	Flexible Manufacturing Systems	Timed Petri Nets
2001	*Zha et al.*	Modeling, Performance Analysis	Robotic Flexible Assembly Systems	Knowledge Petri Nets

Early Petri net studies on manufacturing systems were focused on general problems such as modeling, scheduling, and etc. Today's industrial world includes various manufacturing systems having complex problems. An area of research is improving and developing new methods or techniques for the problems in new manufacturing systems, for example problems in integrated flexible demanufacturing system. Also new problems require new Petri net tool therefore Petri net tools are developed for these problems. Also these problems are solved with new modeling and analyzing methods, which are combined with Petri nets to develop new Petri net class, for example Artificial Intelligence, Neural Network, Expert Systems, Object Oriented, and etc.

3.6 Conclusion

Brief information on Petri nets and Petri net applications in manufacturing systems were given. Following chapter will focus on solving algorithms for SALBP-1 based Petri net.

CHAPTER FOUR
SIMPLE ASSEMBLY LINE BALANCING
A PETRI NET APPROACH

4.1 Introduction

This chapter involves Petri net approach to Simple Assembly Line Balancing Problem-Type I. Five heuristic algorithms based on Petri nets were proposed. First algorithm uses reachability analysis method. Second, third and fourth algorithms employ *T*-invariants, and last algorithm is based on *P*-invariant analysis. These algorithms will be explained by an example.

4.2 Using Petri Nets for SALBP-1

Petri nets can be applied informally to any area or system that can be described graphically like flow charts and that needs some means of representing parallel or concurrent activities (Murata, 1989). An assembly system includes parallel and concurrent activities, and precedence diagram of its tasks is represented as a flow chart. Therefore assembly systems have been one of the popular application areas for Petri net researchers, especially for modeling, designing, and controlling the system and for sequencing jobs (or tasks) on the assembly line (Teng & Zhang, 1993; Cao & Sanderson, 1994; McCarragher, 1994; Thomas *et al.*, 1996; D'Souza & Khator, 1997; Ramaswamy *et al.*, 1997; Adamou *et al.*, 1998; Moore *et al.*, 1998 and 2001; Zha *et al.*, 1998; Jeng *et al.*, 1999; Yee & Ventura, 1999; Zimmermann & Hommel, 1999; Zussman & Zhou,1999; Frey, 2000; Yeung & Moore, 2000; Zha, 2000; Zha & Lim, 2000; Tang *et al.*, 2001; Zha *et al.*, 2001). These studies were given in literature review section in chapter three. We saw that much research has been devoted to assembly systems' problems. Despite these efforts, Petri nets have not been used to solve assembly line balancing problem.

We have two reasons to use Petri nets for SALBP-1. First, an assembly line contains parallel and concurrent activities, and precedence diagram can be easily modeled using Petri nets. Second is related with usage of Petri nets for generating candidates in stage of determination and selection of tasks for a station. In literature, branch and bound procedures, tabu search metaheuristics, genetic algorithms, simulated annealing, and expert systems are used for this generation process. Reachability property of Petri nets can be used for this aim, and Petri nets also allow to dynamic study via token movement. Dynamic study is very helpful to generate candidate tasks and to select the task to be assigned (allocated) among the candidates.

Before the explanation of the algorithms, notations used in these algorithms and their meanings will be given in the next section.

4.3 Notations in the Algorithms and Their Meanings

Developed algorithms will be given in the following sections. Notations in the algorithm and their meanings are given alphabetically. All of them may not be into each of all algorithms.

A: Incidence matrix (*tn* x *pn*)

a: A control variable. Prevents extra assignments to a station under the same token condition.

AL: A vector includes the information related with temporary assigned task. Such as alternative route's number, station number, task number, task time, station time, and idle time.

alc: Number of columns of *ALS*

alr: Number of rows of *ALS* (or number of routes)

ALS: A matrix of *AL* vectors created for one assignment stage by searching all routes.

ASW: A vector representing the assigned task numbers *awr* : Number of rows of *ASW* (or number of assigned tasks)

B: A vector representing j_{th} row in *A*, when *j* is the enabled transition number

Bz: Idle time of station for each available task (for each candidate to be assigned)

C: Cycle time

cr: Number of available tasks to assign (in other words, number of rows of *CJ*)

CJ: A matrix including *CN* vectors

CN: A vector representing the available task number and the task order in the sequence to assign

d: A vector including the transition numbers in each subnet

F: A matrix including the firing sequence of transitions according to the *P*-invariants (in other words, each column in *F* represents alternative routes or subnets of transitions)

fc: Number of columns of *F* (or number of subnets, or alternative routes)

GT: A matrix including the transition numbers which still continues to firing

h: A vector including the place numbers in each *P*-invariant

IST: A vector including the last column of *ALS* (in other words, it represents the idle times of alternative routes)

JN: Selected task number from candidates (*CJ*)

JS: Selected task order from the candidates (available tasks)

LE: Line efficiency of the assembly line

M: A vector representing the initial token condition (1 x *pn*)

minBz: Minimum idle time of station

mint: Minimum firing completion time into alternative firings

MN: A vector, selected from *MY* matrix according to the selected task (1 x *pn*)

MT: A vector including the number of tokens in input places of any transition

MY: A matrix including *B* vector for each enabled transition

nsr: Number of unassigned tasks (in other words, number of rows of *NSEQ*)

NSEQ: A vector representing the firing sequences of unfired transitions, and it is updated continuously during the assignment stage

O: A matrix including all *Q*

Ot: Time of available task (enabled transition or candidate)

Oz:	Candidate (available task or enabled transition) number
P:	*P*-invariants of Petri net as a matrix
pc:	Number of columns of *P*
pn:	Number of places in the net (or number of columns in *A*)
pr:	Number of rows of *P*
Q:	A vector including enabled transition (available task or candidate) number and its firing time
SEQ:	A vector representing the firing sequence of transitions
SN:	A vector including information about tasks that can be assigned (that is, SN includes station number (*w*), assigned task number (*Oz*), assigned task time (*Ot*), idle time of station (*minBz*))
SON:	A matrix including information about assigned (allocated) tasks (that is, it includes station number (*w*), assigned task number (*Oz*), assigned task time (*Ot*), idle time of station (*minBz*))
sr:	Number of sequenced tasks
s1:	Number of rows of *GT* (in other words, number of transitions which still continues to firing)
SumOt:	Total times of allocated (assigned) tasks
Sumt:	Total times of tasks (transitions)
Sw:	Total times of assigned tasks in a station (or station job time)
Swnow:	Station time used in alternative routes (or temporary station time)
t:	A vector representing the time of tasks (or transitions) (1 x *tn*)
time:	Current time before firing of any transition
tn:	Number of transitions in the net (or number of rows in *A*)
vr:	Number of tasks (or transitions) that can be assigned (or can fire) (in other words, *vr* is the number of the rows of *O* matrix)
w:	Station number
ww:	Station number used in alternative routes (or temporary station number)
x:	Minimum idle time into *IST*
yg:	A control value is to determine any transition which still continues to firing
yt:	Firing completion time of any enabled transition

z: number of route giving minimum idle time (or row number of *x* in *IST* vector)

4.4 Petri Net-Based Solving Algorithms for SALBP-1

All five proposed algorithms would be given in the following subsections in details. In these algorithms, first work is to transform precedence diagram of tasks to a Petri net model where tasks are represented by transitions. Important point of this step is that task number and transition number must be the same. For example, first task must be represented by the first transition. This is important from the point of view of analyzing the results of algorithms. Thereby Petri net model includes number of transitions which is equal to number of tasks in the line. Also number of places in the net is equal to sum of the number of precedence relations in the precedence diagram and number of tasks, which haven't got any predecessor task or successor task in the precedence diagram. There are three type of places in the Petri net model of the precedence diagram; first is the source places, which have no input transition, second is places, which have one input and one output transitions, and third is sink places, which have no output transition. For example, if a line balancing problem includes 10 tasks, one of them which hasn't any predecessor task will be source place, two tasks which haven't any successor task will be sink places, and each of 12 precedence relations will be represented by one place which has one input and one output transitions. Therefore, Petri net model will have 10 transitions and 15 places. This information is helpful for verification of the Petri net model. Second work is to determine the incidence matrix of the Petri net model. Incidence matrix is used in all dynamic study stages to find enabled transitions and to form new token condition in the net. Incidence matrix represents the relation between places and transitions. Therefore number of its rows and columns are equal to number of transitions and number of places, respectively. Another work is to design vector *M* representing the initial token condition. This vector includes one row and columns, whose number is equal to number of places in the net. Other works are to create a vector of time of tasks (or transitions) and to decide a cycle time. Last algorithm also needs *P*-invariants to be computed from the incidence matrix.

In the followings, each proposed algorithm is presented in a subsection.

4.4.1 Solving SALBP-1 with Reachability Analysis (RA)

The aim is to minimize the number of stations on the line according to a given cycle time. This algorithm combines reachability analysis method with a decision rule that is to select the task given the minimum idle time. It contains two main steps; first step is to collect inputs, and second step is to use these inputs to solve SALBP-1. Steps of algorithm are given below. Also flow chart of the algorithm is given in figure 4.1.

Step 1. *Input A, M, t, and C*
Determine pn and tn using A, determine Sumt
Initial values: w=1, Sw=0, Bz=C, MinBz=C, SumOt=0, MT=[], MY=[], MN=[], Ot=0, Oz=0, O=[], SN=[], SON=[], a=0
Step 2. *While there exist unassigned tasks (Sumt>SumOt) do*
a. *Determine enabled transitions (or tasks)*
for j=1, 2, ... ,tn
i. *for i=1,2,...,pn*
if A(j,i)=-1
do y=M(1,i),
Update MT by adding y to MT
ii. *if all values in MT is equal to 1*
do B=A(j,:)
Update MY by adding B to MY
Q=[j t(j)]
Update O by adding Q to O
iii. *MT=[]*
b. ***Select a task that makes idle time minimum, and assign this task to the station.***
Determine vr
for v=1,2,...vr
i. ***if O(v,2)=C-Sw and a≠1***
do minBz=0, MN=MY(v,:), Oz=O(v,1), Ot=O(v,2)
SN=[w Oz Ot minBz], a=1
ii. ***if O(v,2) < C-Sw and a≠1***
do Bz=C-Sw-O(v,2)
aa. ***if Bz<minBz and Bz>0***
do minBz=Bz, MN=MY(v,:), Oz=O(v,1), Ot=O(v,2),
SN=[w Oz Ot minBz]
c. ***Update SON by adding SN to SON***
tim=SON(:,3)
SumOt=∑tim; Sw=C-minBz
i. ***if Oz=[]***

do w=w+1; Bz=C; minBz=C; Sw=0; MN is a zero vector (1xpn)
M=M+MN; MN=[]; MY=[]; O=[]; Oz=0; Ot=0; B=[]; SN=[]

Step 3. *Show the results (SON matrix)*
Find line efficiency, LE=Sumt/(wxC)
Show the line efficiency (LE)

A is the incidence matrix of the Petri net model, and *pn* and *tn* are the number of places and the number of transitions, respectively. Initial token condition (*M*) represents the required minimum token number for firing all transitions. Task times are given in vector *t*. Initial values are assigned to the variables. Thus, station number *w* is 1, station time *Sw* is 0, idle time *Bz* and minimum idle time *minBz* are equal to cycle time *C*, total of assigned task times *SumOt* is 0, matrices; *MT*, *MY*, *O*, and *MN* which are used in determining candidates, are initially empty. Also matrices *SON* and *SN*, which include the results in the end of this algorithm and results of each assignment, respectively, are empty. Since initially there isn't any determined candidate, *Ot* and *Oz* are 0. *a* is 0 because there isn't any assignment.

In the second step, the task giving minimum idle time is assigned to station. Therefore, firstly candidates are determined, a task making idle time minimum is selected from available tasks and assigned to the station, and then variables are updated for new condition.

Determining the available tasks is determination of the enabled transitions. Incidence matrix and initial token condition provide related information (*Step 2.a.*). First of all, input places of transitions are found using incidence matrix (*Step 2.a.i.*). Searching starts in the first row of *A* for first transition. The element corresponding to input place *p* of transition *t* will be –1. All column numbers, which include –1 value in the row, are determined as the input place number of the transition, and then token conditions of determined input places are searched using token condition vector *M*. +1 value in the columns of *M* means that the transition is enabled (*Step 2.a.i.i*). This transition number and its firing time are placed into vector *Q* as if to fire. *O* matrix includes all *Q*s. Also the related row is stored in *MY* matrix to use in updating of token condition. These operations are repeated until all transitions are searched. Therefore all firable transitions (i.e. available tasks to be assigned to station) are determined. End of the each searching stage, vector *MT* is made empty (*Step 2.a.ii.i*).

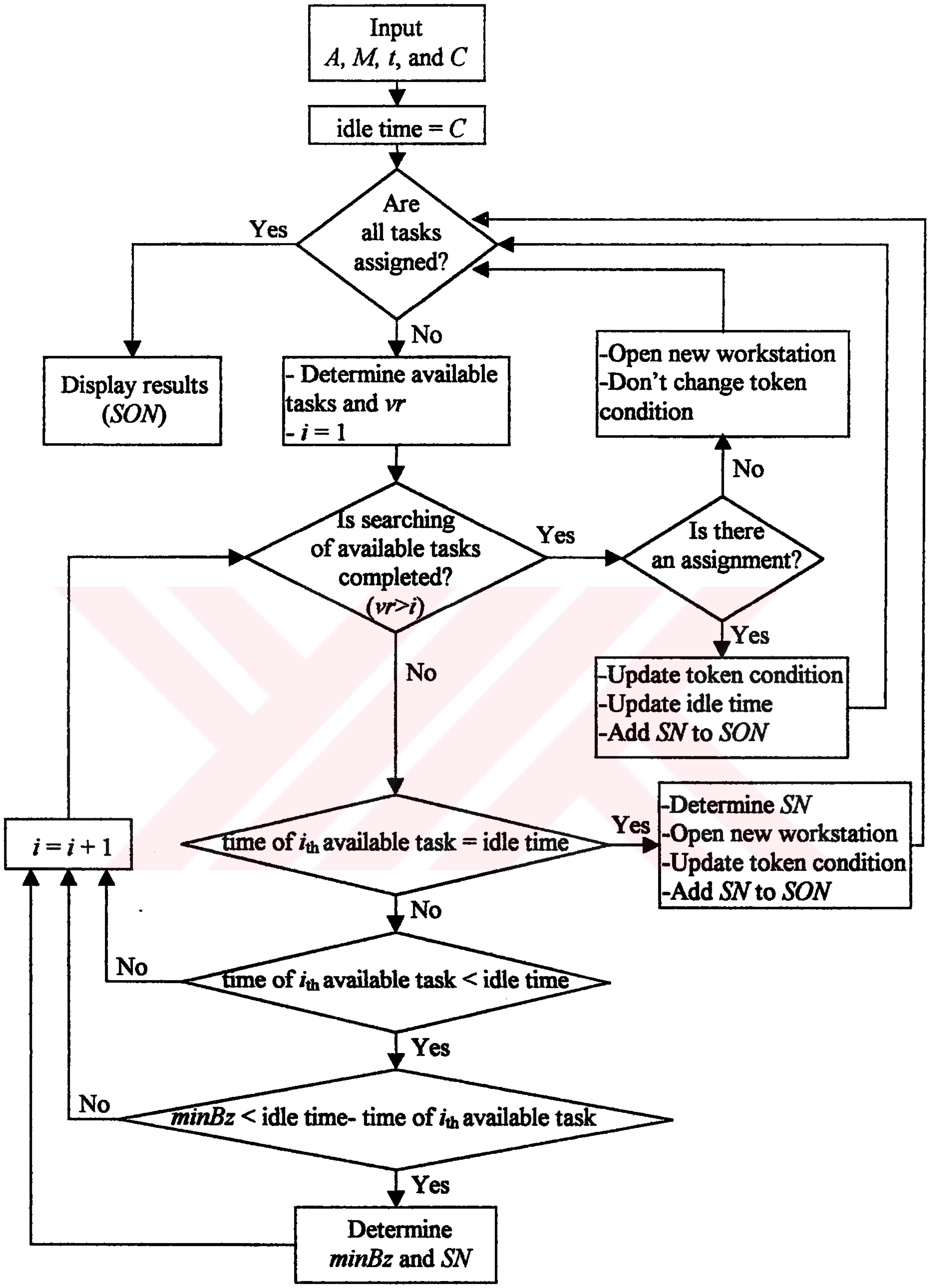


Figure 4.1 Flow chart of the RA algorithm

After the available tasks were determined, transitions are selected from them and assigned to the station in the second sub step (*Step 2.b.*). Inputs of this sub step are *O* matrix, vector *t* that represents task times (i.e. transition firing times), and cycle time (*C*). *O* matrix includes available tasks (i.e. transitions). Therefore number of rows of *O* (*vr*) gives the number of candidates. *a* is the control variable to prevent more than one assignment process under same token condition. In this sub step, if the transition's firing time is equal to idle time of station and there isn't any assignment for current token condition (a=0) then the task is assigned to the station immediately (*Step 2.b.i.*). If there is any transition providing this condition, then this transition (i.e. task) is selected and assigned to the station. After this assignment, value of *a* changes from 0 to 1. Therefore, under the same conditions, another task assignment to the station is not allowed. Station number, task number, task time, and idle time are placed into vector *SN* as a result of assignment step. Also *MN* vector is determined into *MY* matrix to determine the new token condition. If there isn't any task providing the zero idle time and there isn't assigned task (a=0) then the task given the minimum idle time is selected from candidates and assigned to the station. This is done while task time *O*(*v*,2) is less than the current idle time (*C-Sw*) (*Step 2.b.ii.*). Candidate idle times are determined for each available task (*Step 2.b.ii.aa.*). First *Bz* value is less than *minBz*, because initial value of *minBz* is equal to the cycle time. Therefore first *Bz* is *minBz*. If another candidate provides less idle time (*minBz*) than first candidate, then its Bz will be *minBz*. This search is done for all available tasks. After a task giving minimum idle time is selected, vectors *MN* and *SN* are created. Control variable *a* prevents extra assignments. In *Step 2.c.*, vector *SN* is added to matrix *SON*, and total time of assigned tasks (*SumOt*) is determined using 3rd column of this matrix, and current station time *Sw* is calculated by subtracting current idle time (*minBz*) from cycle time. If candidates are not appropriate for current station, any assignment does not occur, that is *Oz*=[], therefore new station should be opened to allocate (assign) the task (*Step 2.c.i.*). In this case, station number increases by one (w=w+1), and idle time *Bz*, minimum idle time *minBz*, and total of task times in station *Sw* are made equal to their initial values. Furthermore, since there isn't any assignment to current station, token condition in the net mustn't change, that is, elements of vector *MN* are zero. After this step, adding the *MN* (elements of this row will be zero if any assignment was not made) to current token condition vector gives the new token condition vector (*Step*

2.c.ii.). For new operations, all vectors and matrices; *MN*, *MY*, *O*, *B*, and *SN*, which are used in selecting and allocating stages, are emptied. Selected and assigned task number *Oz* and task time *Ot* are also 0 for the new assignment stage. These operations are repeated until all tasks are allocated to stations (while Sumt>SumOt). When the all tasks are allocated to stations (Sumt=SumOt) *Step 3* begins.

Findings are given in the last step. During the *Step 2*, *SON* matrix is continuously updated by adding information about each allocation (assignment) *SN*. Therefore *SON* matrix is ready to show the results when the *Step 2* finishes. Station number, task number, task time, and current idle time of related station are given in the *SON* matrix. Line efficiency is calculated by dividing total of tasks times *Sumt* to number of stations (last value of *w*).

4.4.2 Solving SALBP-1 with Firing Sequences of Transitions (FS)

Second algorithm uses the firing sequence of transitions to allocate the tasks to stations. This algorithm contains two main steps; first is to create a firing sequence, second is to allocate the tasks by using this firing sequence. Algorithm is given below. Flow chart of the algorithm is presented in figure 4.2.

Step 1. *Input A, M, t, and C.*
Determine pn and tn using A. Determine Sumt.
Initial values: w=1, Bz=C, time=0, mint=Sumt+1, SumOt=0, MT=[], MN=[], Oz=0, O=[], SEQ=[], NSEQ=[], SN=[], SON=[], MB=M, sr=0, GT=[0], yg=0
Step 2. *While all transitions are sequenced (tn > sr) do*
- ***a.*** *Determine transitions (or tasks) that are enabled (or can be assigned to station)*
  for j=1, 2, ... ,tn
  - ***i.*** *for i=1,2,...,pn*
    if A(j,i)=-1
    do y=M(1,i)
    Update MT by adding y to MT
  - ***ii.*** *if all values in MT is equal to 1*
    do g=j and determine s1 using GT
    - ***aa.*** *for s=1,2,..s1*
      - ***aaa.*** *if g=GT(s)*
        do yg=1
    - ***bb.*** *if yg≠1*
      do yt=time+t(j) and Q=[j yt]

Update O by adding Q to O
iii. MT=[] and yg=0
b. Determine firing sequence of transitions
Determine vr
for v=1, 2,..., vr
i. if O(v,2) < mint
do mint=O(v,2), Oz=O(v,1), and r=v
ii. time=mint, MN=A(Oz,:), M=M+MN, mint=Sumt
Update SEQ by adding Oz to SEQ
Determine the sr
Update O by deleting number of transition (Oz) which is added to SEQ
iii. if O=[]
do GT=[0]
else
do GT=O(:,1)

Step 3. *Transitions are assigned according to the firing sequence*
NSEQ = SEQ, M=MB, MT=[]
While there exist unassigned tasks (Sumt>SumOt) do
a. if Bz =0 or Bz<t(NSEQ(1,:))
do Bz=C, w=w+1, SN=[]
b. if Bz>=t(NSEQ(1,:))
do Bz=Bz-t(NSEQ(1,:)), SumOt=SumOt+t(NSEQ(1,:))
M=M+A(NSEQ(1),:), SN=[w NSEQ(1,:) t(NSEQ(1,:)) Bz];
Determine nsr
Update NSEQ by deleting first row from NSEQ
c. Update SON by adding to SN to SON

Step 4. *Show the results (SON matrix)*
Find line efficiency, LE=Sumt/(wxC)
Show the line efficiency (LE)

Like in the first algorithm, Petri net model of the precedence diagram is built and the initial values are decided in *Step 1*. Firing sequence is found out in *Step 2*. Enabled transitions and input places of each transition are determined in *Step 2.a.* and in *Step 2.a.i.*, respectively. Candidate transitions, which didn't start to fire (*yg*=0), and their firing completion times are listed in a matrix (*O*). Firing sequence of transitions is found in *Step 2.ii.bb.* If any transition started to fire in previous searching and it is still firing, then its firing completion time does not change. In *Step 2.a.ii.aa.aaa.*, yg is determined as 1 for these transitions. Thus *yg* keeps firing information of these transitions (*Step 2.a.ii.aa.* and *Step 2.a.ii.bb*). In *Step 2.iii.*, *matrix MT* is emptied , and *yg* is made equal to zero for new search. Task, which has minimum firing time, is selected as a first task. (*Ste p2.b.i.*), in *Step 2.b.ii.*, variables are updated considering this selection, and the firing completion instant of this transition is determined as current time. New token condition is calculated by adding related row of incidence matrix to current condition. Selected transition number is added to sequencing list *SEQ*, and is deleted from candidate transition list *O*. Other candidates and their firing completion times are kept in the list, because they are firing when the selected transition completes its firing (*Step 2.b.iii.*). *Step 2* is repeated for new candidates, which are determined in *Step 2.a.*, and is repeated for old candidates, which remains from previous processes. Determining candidates and selecting among them are repeated until all transition numbers placed into sequence list.

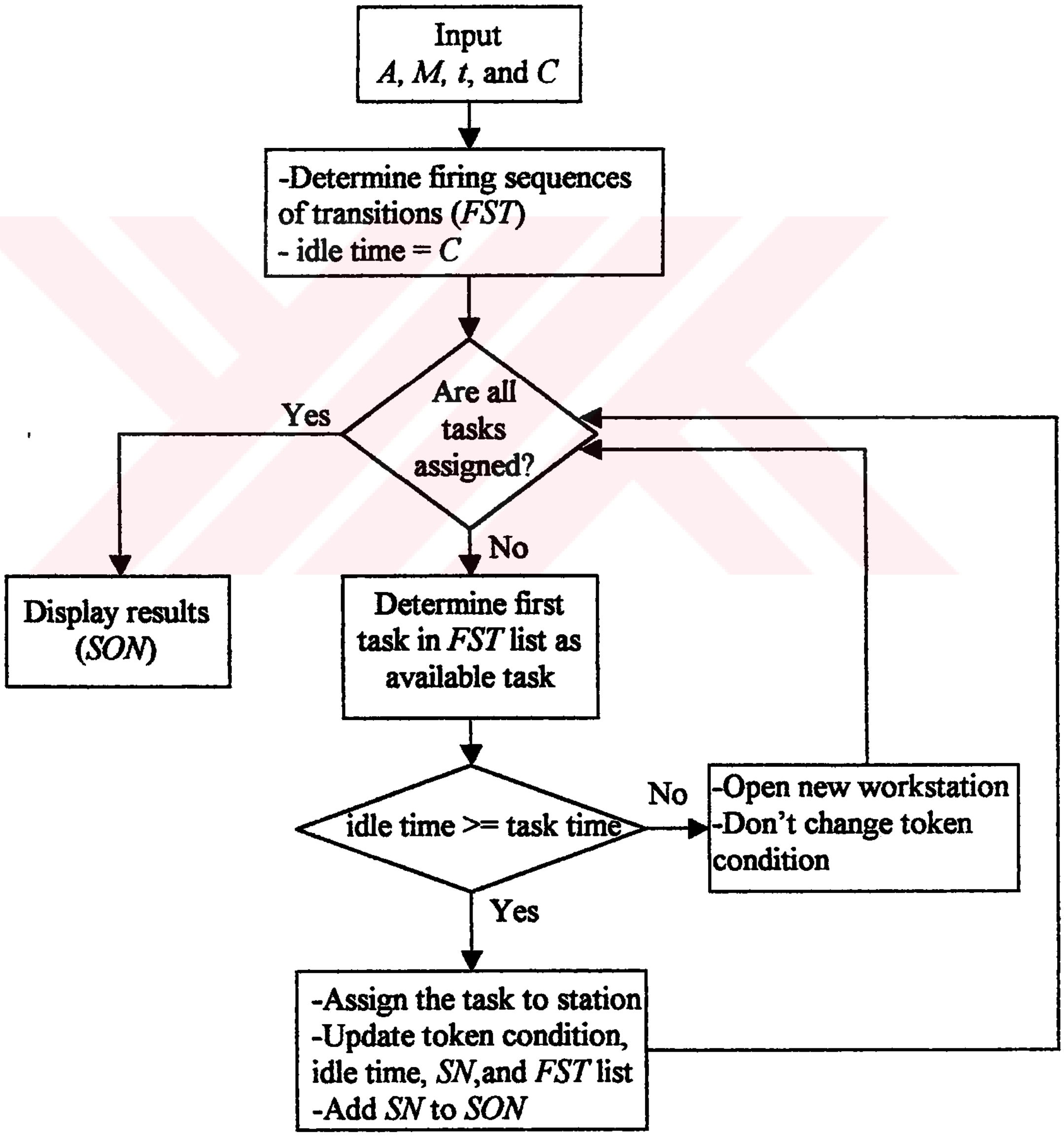


Figure 4.2 Flow chart of the FS algorithm

In *Step 3*, tasks (or transitions) are allocated to station using sequence list made in *Step 2*. New sequence *NSEQ* is created to use in all allocation stage. In the beginning, *NSEQ* is equal to sequence list *SEQ*. During the allocation, *NSEQ* is updated by deleting the selected task number from *NSEQ*. There are two alternatives during the allocation process (*Step 3.a.* and *Step 3.b.*). If the current idle time of station is equal to zero or current idle time is less than the first task's time in the sequence, station is full or not available and new station could be opened (*Step 3.a.*). This alternative is finished by updating, that is, idle time is made equal to cycle time, station number is increased by one, and all information in *SN* are deleted. In the second alternative, if first task's time is equal or greater than the idle time, first task in the sequence list is allocated to station (*Step 3.b.*). After this, new token condition *M*, current idle time of station *Bz*, information related with selected task *SN*, number of rows of *NSEQ*, and new sequence *NSEQ* are updated. In *Step 3.c.*, *SON* matrix is enlarged by adding *SN*. All processes in *Step 3* are repeated until new sequence list becomes empty (while *Sumt>SumOt*).

When the new sequence list becomes empty (*Sumt=SumOt*), all tasks on the line are allocated to the stations. *Step 4* shows which task is placed into which station. Line efficiency is also calculated and displayed as a result.

4.4.3 Solving SALBP-1 with Firing Sequences of Transitions-First Search (FS-f)

This algorithm and second algorithm are very like. Second algorithm uses firing sequences of transitions to solve SALBP-1. Tasks are allocated using this firing sequence and cycle time information. If firing time of a transition in the sequence is greater than the current idle time of the station then new station will be opened to allocate this task. First search algorithm uses the same firing sequence, but it doesn't open new station immediately when the idle time is not available to allocate the task in the sequence. The new algorithm determines the available tasks to be assigned (or transitions to be fired). These candidates are arranged in an order like in the firing sequence. First task in the list, whose time is equal to or less than the current idle time of the station, is assigned to the station. If any candidate cannot be assigned to the current station then a new station is opened, and allocation starts with the first task in the sequence or with a task, whose time is greater than the idle time of previous station. Here, this step increases the second algorithm's efficiency. The algorithm is given below. Also flow chart of the algorithm is shown in figure 4.3.

Step 1. *Input A, M, t, and C.*
Determine pn and tn using A. Determine Sumt.
Initial values: w=1, Bz=C, time=0, mint=Sumt+1, SumOt=0, MT=[], MN=[], Oz=0, O=[], SEQ=[], NSEQ=[], SN=[], SON=[], MB=M, sr=0, GT=[0], yg=0, CJ=[],a=0

Step 2. ***While all transitions are sequenced (tn > sr) do***
- ***a.*** ***Determine transitions (or tasks) that are enabled (or can be assigned to station)***
  for j=1, 2, ... ,tn
  - ***i.*** ***for i=1,2,...,pn***
    if A(j,i)=-1
    do y=M(1,i)
    Update MT by adding y to MT
  - ***ii.*** ***if all values in MT is equal to 1***
    do g=j and determine s1 using GT
    - ***aa.*** ***for s=1,2,..s1***
      - ***aaa.*** ***if g=GT(s)***
        do yg=1
    - ***bb.*** ***if yg≠1***
      do yt=time+t(j) and Q=[j yt]
      Update O by adding Q to O
  - ***iii.*** ***MT=[] and yg=0***
- ***b.*** ***Determine firing sequence of transitions***
  Determine vr
  for v=1, 2,..., vr
  - ***i.*** ***if O(v,2) < mint***
    do mint=O(v,2), Oz=O(v,1), and r=v
  - ***ii.*** ***time=mint, MN=A(Oz,:), M=M+MN, mint=Sumt***
    Update SEQ by adding Oz to SEQ
    Determine the sr
    Update O by deleting number of transition (Oz) which is added to SEQ
  - ***iii.*** ***if O=[]***
    do GT=[0]
    else
    do GT=O(:,1)

Step 3.Transitions are assigned according to the firing sequence
NSEQ = SEQ, M=MB, MT=[]
While there exist unassigned tasks (Sumt>SumOt) do
- ***a.*** ***if Bz = 0***
  do Bz=C, w=w+1, SN=[]
- ***b.*** ***if Bz<t(NSEQ(1,:))***
  Determine nsr

i. for n=1,2,...,nsr
aa. for i=1,2,...,pn
aaa. if A(NSEQ(n),i)=-1
do y=M(1,i)
Update MT by adding y to MT
bbb. if all values in MT are equal to 1
do CN=[n NSEQ(n)]
Update CJ by adding CN to CJ
bb. MT=[]
ii. Determine cr
aa. for cc=1,2,...,cr
aaa. if Bz>=t(CJ(cc,2)) and a≠1
do Bz=Bz- t(CJ(cc,2)), SumOt=SumOt+ t(CJ(cc,2))
SN=[w CJ(cc,2) t(CJ(cc,2)) Bz], MN=M+A=(CJ(cc,2)),:),
a=1
Update NSEQ by deleting assigned task number (CJ(cc,2))
from NSEQ
iii. if CJ=[] or a≠1
do Bz=C, w=w+1, SN=[]
c. if Bz>=t(NSEQ(1,:)) and a≠1
do Bz=Bz-t(NSEQ(1,:)), SumOt=SumOt+t(NSEQ(1,:))
M=M+A(NSEQ(1),:), SN=[w NSEQ(1,:) t(NSEQ(1,:)) Bz];
Determine nsr
Update NSEQ by deleting first row from NSEQ
d. Update SON by adding to SN to SON
CJ=[] and a=0
***Step 4**. Show the results (SON matrix)*
Find line efficiency, LE=Sumt/(wxC)
Show the line efficiency (LE)

Except *Step 3* all steps of this algorithm are the same as second algorithm. If idle time is equal to zero then new station is opened and idle time is updated as circle time (*Step 3.a.*). If idle time is less than the time of task in the sequence then availability of next tasks in the sequence are searched (*Step 3.b.*). Availability of all tasks is investigated using incidence matrix and current initial token condition (*Step 3.b.i.aa.aaa.*). Available tasks are listed in matrix *CJ* as candidates (*Step 3.b.i.aa.bbb.*). Times of candidates are compared with the idle time of the station. (*Step 3.b.ii.aa.aaa.*). First task in the candidate list, whose time is less than the idle time, is allocated to the current station. *SN* vector is created, new token condition is found and *NSEQ* is updated. Also control variable's (*a*) value is made 1 to prevent undesirable task allocation under the same token condition. *SN* is added to *SON* in *Step 3.d.* If any allocation cannot be made to the current station, then a new station is opened (*Step 3.b.iii.*). New allocation is done beginning from the first task in the sequence (*NSEQ*) (*Step 3.c.*). Other steps are the same as previous algorithm.

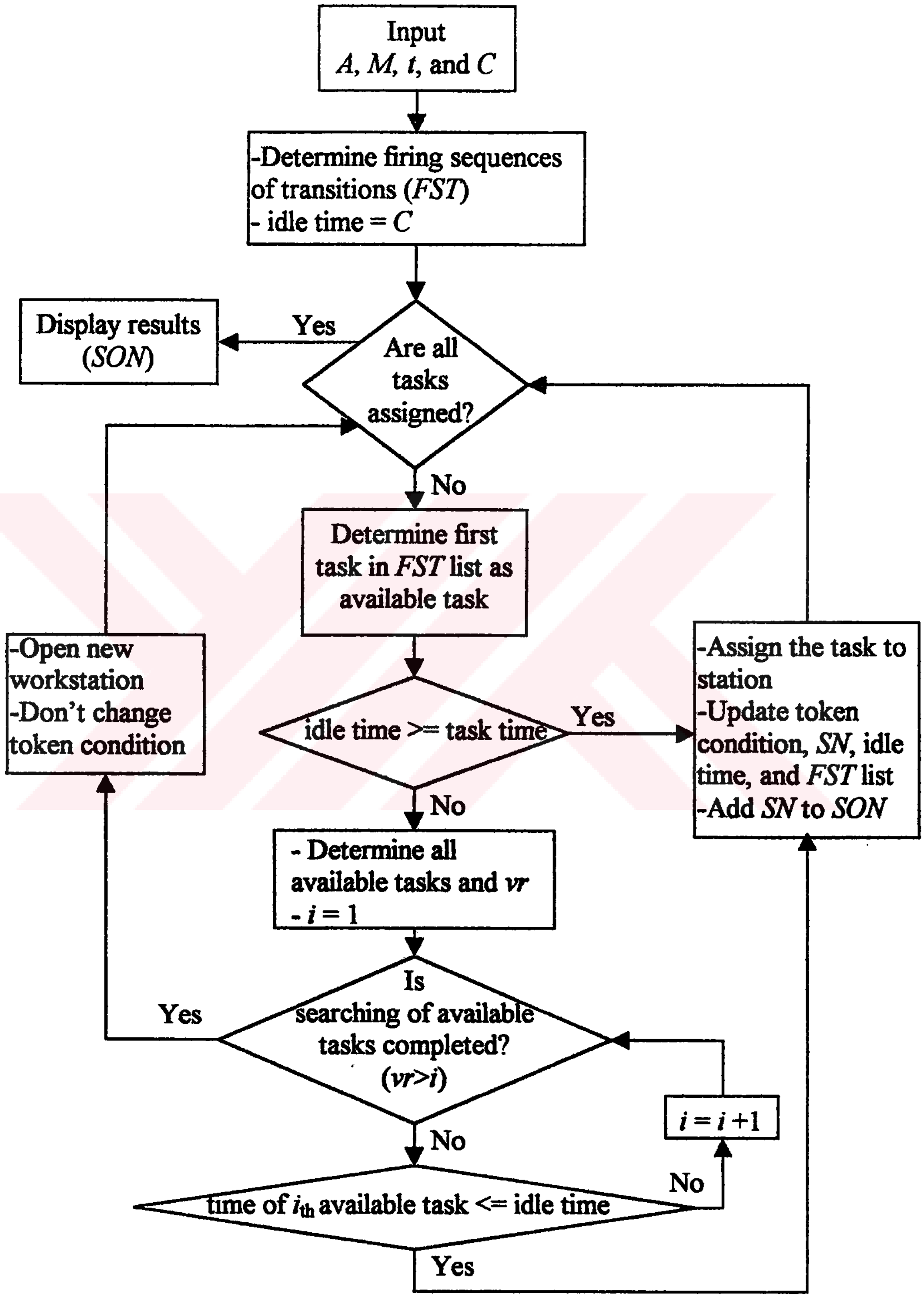


Figure 4.3 Flow chart of the FS-f algorithm

4.4.4 Solving SALBP-1 with Firing Sequences of Transitions-Best Search (FS-b)

Main difference between this algorithm and the previous algorithm First Search is that the task giving minimum idle time for the current station is considered as a candidate and then assigned to the current station instead of the first available task in the sequence, if time of the first task is greater than the idle time of the current station. Firing sequences are used like in the previous two algorithms. In this algorithm, candidate tasks are determined like in the previous algorithm, and then the task given minimum idle time is assigned to the station. Steps of this algorithm are as follows. Flow chart of the algorithm is presented in figure 4.4.

Step 1. *Input A, M, t, and C.*
Determine pn and tn using A. determine Sumt.
Initial values: w=1, Bz=C, time=0, mint=Sumt+1, SumOt=0, MT=[], MN=[], Oz=0, O=[], SEQ=[], NSEQ=[], SN=[], SON=[], MB=M, sr=0, GT=[0], yg=0, CJ=[], a=0, minBz=C, Sw=0, JN=0, JS=0
Step 2. *While all transitions are sequenced (tn > sr) do*
a. *Determine transitions (or tasks) that can fire (or can be assigned to station)*
for j=1, 2, ... ,tn
i. *for i=1,2,...,pn*
if A(j,i)=-1
do y=M(1,i)
Update MT by adding y to MT
ii. *if all values in MT is equal to 1*
do g=j and determine s1 using GT
aa. *for s=1,2,..s1*
aaa. *if g=GT(s)*
do yg=1
bb. *if yg≠1*
do yt=time+t(j) and Q=[j yt]
Update O by adding Q to O
iii. *MT=[] and yg=0*
b. *Determine firing sequence of transitions*
Determine vr
for v=1, 2,..., vr
i. *if O(v,2) < mint*
do mint=O(v,2), Oz=O(v,1), and r=v
ii. *time=mint, MN=A(Oz,:), M=M+MN, mint=Sumt*
Update SEQ by adding Oz to SEQ
Determine the sr
Update O by deleting number of transition (Oz) which is added to SEQ
iii. *if O=[]*
do GT=[0]

else
do GT=O(:,1)

Step 3. *Transitions are assigned according to the firing sequence*
NSEQ = SEQ, M=MB, MT=[]
While there exist unassigned tasks (Sumt>SumOt) do
a. *if Bz = 0*
do Bz=C, w=w+1, SN=[], minBz=Bz
b. *if Bz<t(NSEQ(1,:))*
Determine nsr
i. *for n=1,2,...,nsr*
aa. *for i=1,2,...,pn*
aaa. *if A(NSEQ(n),i)=-1*
do y=M(1,i)
Update MT by adding y to MT
bb. *if all values in MT are equal to 1*
do CN=[n NSEQ(n)]
Update CJ by adding CN to CJ
cc. *MT=[]*
ii. *Determine cr*
aa. *for cc=1,2,...,cr*
aaa. *if Bz>=t(CJ(cc,2))*
aaaa. *if minBz>=C-Sw-t(CJ(cc,2))*
do minBz=C-Sw- t(CJ(cc,2)), JN=CJ(cc,2),
JS=CJ(cc,1), a=1
iii. *if CJ=[] or a≠1*
do Bz=C, w=w+1, SN=[]
iv. *else if CJ=[] or a≠1*
do Bz=Bz-t(JN), SumOt=SumOt+t(JN), minBz=Bz
SN=[w JN t(JN) Bz], M=M+A(JN,:)
Update NSEQ by deleting JN from NSEQ
c. *if Bz>=t(NSEQ(1,:)) and a≠1*
do Bz=Bz-t(NSEQ(1,:)), SumOt=SumOt+t(NSEQ(1,:)), minBz=Bz
M=M+A(NSEQ(1),:), SN=[w NSEQ(1,:) t(NSEQ(1,:)) Bz]
Determine nsr
Update NSEQ by deleting first row from NSEQ
d. *Update SON by adding to SN to SON*
Sw=C-Bz, CJ=[], and a=0

Step 4. *Show the results (SON matrix)*
Find line efficiency, LE=Sumt/(wxC)
Show the line efficiency (LE)

Until *Step 3*, all operations in this algorithm are the same as in the previous two algorithms. In this algorithm, four new variables; *minBz*, *JN*, *JS*, and *Sw* are used and defined in *Step 1*. When the allocation process starts, assignment operations are done using *Step 3.c.* until the station's idle time becomes less than the time of first task in the sequence. *MinBz* is made equal to *Bz* in the end of the *Step 3.c.* If the idle time is

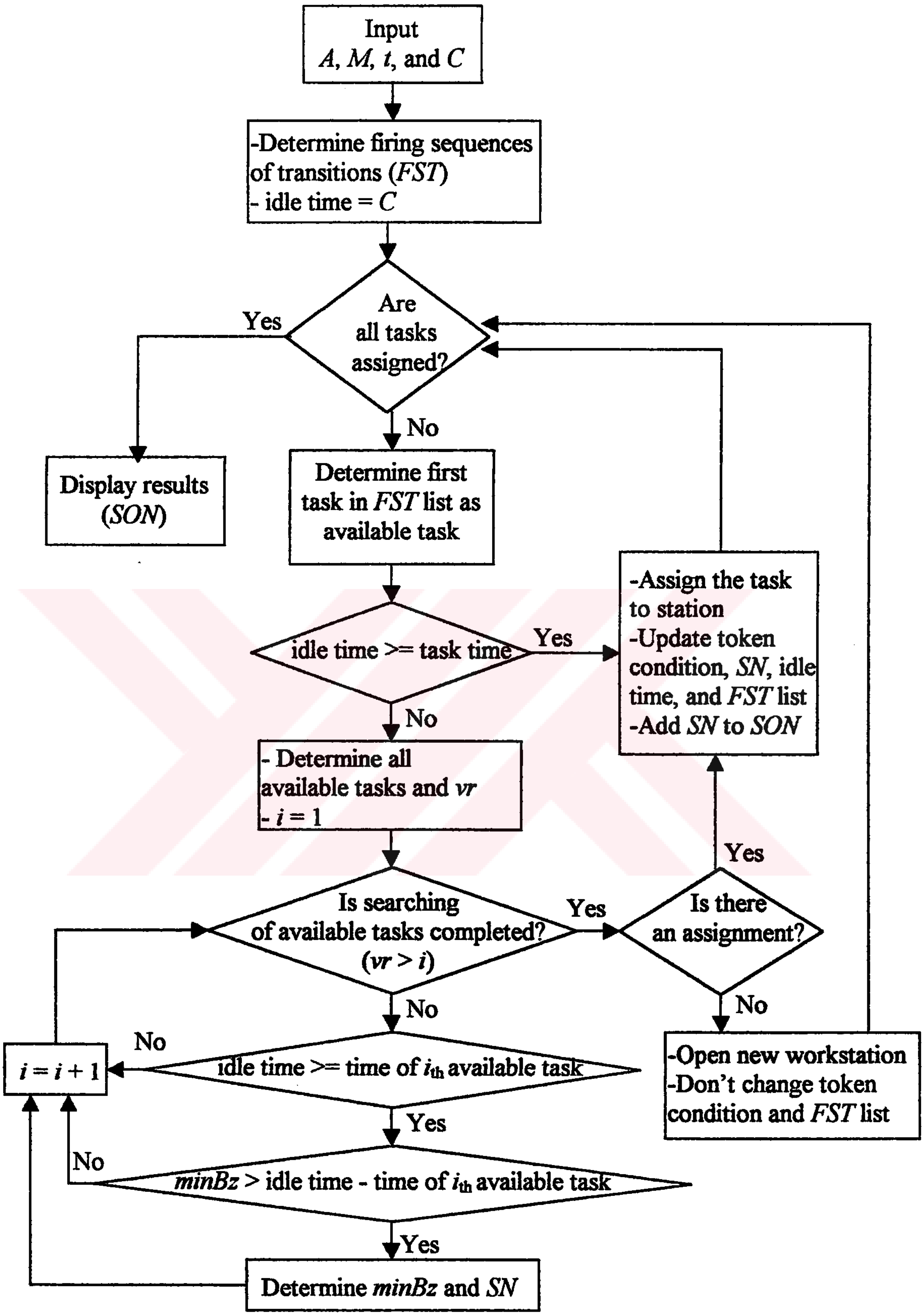


Figure 4.4 Flow chart of the FS-b algorithm

less than the time of the first task, then operations, like in the previous algorithm, are done in *Step 3.b.i.* to create the candidates. In *Step 3.b.ii.aaa.*, candidates having less time than the current station's idle time are searched, and at the end (*Step 3.b.ii.aa.aaa.aaaa)* the task giving minimum idle time is identified among the available tasks. In this step, information related with candidate and assigned task is collected such as task number *JN*, task order in the candidates *JS*, and minimum idle time *minBz*. Value of control variable *a* is changed from 0 to 1. Allocation operations are done in *Step 3.b.iv*. Therefore *Bz*, *SumOt*, and *minBz* are calculated, *SN* is created, new token condition *M* is determined, and *NSEQ* is updated by deleting *JS* from the list of unassigned tasks. If any candidate cannot be offered (*CJ*=[]) or any assignment cannot be made among the candidates (*a*=0), then *Step 3.b.iii.* starts to open a new station. Next steps are the same as in the previous algorithm. At the end of this algorithm *SON* is created, and *Sw* is calculated (*Step 3.d.*).

4.4.5 Solving SALBP-1 with *P*-invariants (*P*-in)

While solving assembly line balancing problems, in addition to cycle time minimization, movements between stations are preferred to be little to prevent high transportation cost and traffic jam. Here, this algorithm realizes this aim by minimizing the number of stations. It uses *P*-invariants of Petri net model as alternative routes. According to the task sequences in these alternative routes, tasks are assigned to the stations in such a way minimizing the idle time. This algorithm is given below. Flow chart of the algorithm is given in figure 4.5.

Step 1. *Input A, M, t, C, and P.*
Determine pn and tn using A, and determine pr and pc using P
Initial values: w=1, Sw=0, MT=[], h=[], d=[], F=[], ALS=[], ASW=[], awr=0, SON=[]
Step 2. *Determine the alternative routes (or subsets of transitions) using P*
for i=1, 2,..., pr
a. *for j=1,2,...,pc*
i. *if P(i,j)=1*
do h=[h;j]
b. *Determine hr using h*
c. *for k=1,2,..hr-1*
i. *for l=1,2,..., tn*
aa. *if A(l,h(k))=-1 and A(l,h(k+1))=1*
do d=[d;l]

d. Update F by adding d to F
h=[], d=[]

***Step 3.** Tasks are allocated to stations using F*
F=F'
Determine fr and fc using F
While there exist unassigned tasks (tn>awr) do
a. for ii=1,2,...,fc
M=MB, Swnow=Sw, ww=w
i. for jj=1,2,...,fr
a=0
aa. for zz=1,2,...awr
aaa. if F(jj,ii)==ASW(awr,1)
do a=1
bb. if F(jj,ii)>0 and a<1do
aaa. for kk=1,2,...,pn
aaaa. if A(F(jj,ii),kk)=-1
do y=M(1,kk)
Update MT by adding y to MT
bbb. if all values in MT is equal to 1, do
aaaa. if C-Swnow>=t(F(jj,ii))
do Swnow=Swnow+t(F(jj,ii))
AL=[ii ww F(jj,ii) t(F(jj,ii)) Swnow C-Swnow]
B=A(F(jj,ii),:), M=M+B
Update ALS by adding AL to ALS
cc. MT=[]
b. if ALS =[]
do w=w+1, Sw=0
c. else if ALS =[]
do IST=ALS(:,6) (last column of ALS)
Determine x and z using IST
Sw=C-x, al=ALS(z,1) (for determining alternative number)
Determine alr and alc using ALS
i. for aa=1,2,...,alr
aa. if ALS(aa,1)=al
Update ASW by adding ALS(aa,3) to ASW
Update SON by adding the new allocation information
MB=MB+A(ALS(aa,3),:)
ii. Determine awr using ASW
ALS=[]

***Step 4.** Show the results (SON matrix)*
Find line efficiency, LE=Sumt/(wxC)
Show the line efficiency (LE)

This algorithm consists of two main steps; in the first step, alternative sequences (or routes) are obtained using *P*-invariant, and in the second step, tasks are allocated to the stations using the sequences.

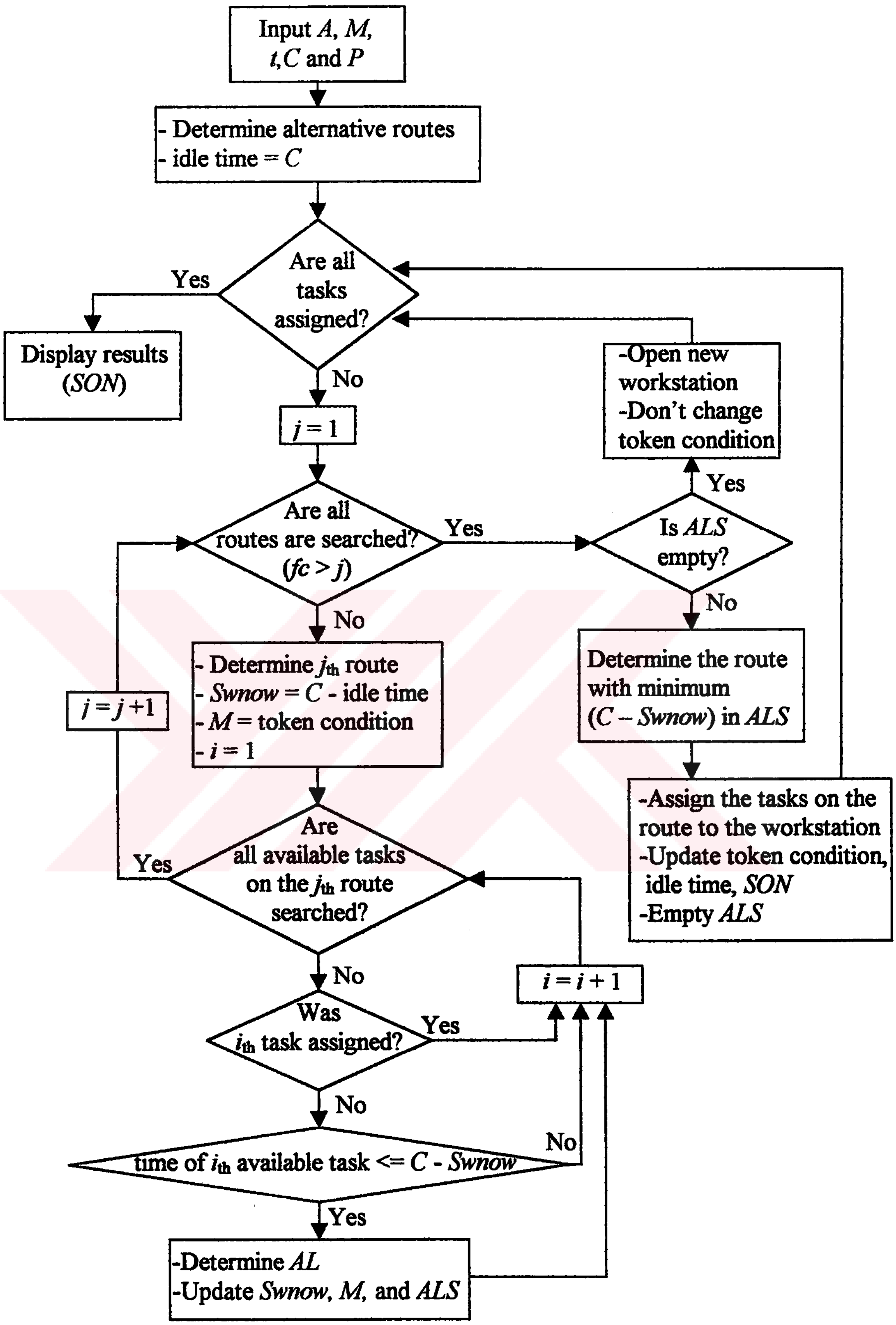


Figure 4.5 Flow chart of the *P*-in algorithm

Related variables and their initial values are determined in the first step like the previous algorithms. Using *P*-invariants of Petri net model makes this algorithm different from the others.

In *Step 2,* alternative routes are determined using *P*-invariants. *P* matrix, which is the input of this algorithm, includes the numbers of places in the *P*-invariants. Successive place numbers in each column of P are listed in a vector ***h*** (*Step 2.a.*), and then number of successive places ***hr*** is determined using ***h*** (*Step 2.b.*). In *Step 2.c.* the sequence of transitions that occupy spaces between places are determined considering the input and output places of transitions using incidence matrix and *P*-invariants. Two successive places in any column of *P* are investigated to find the transition such that these successive places are its input and output places. Transition number is represented by the row number of the incidence matrix and successive place numbers are represented by the column numbers of the incidence matrix. Let j is transition number, and m and n are successive place numbers. If the value in j_{th} row and m_{th} column of the incidence matrix is equal to –1, then m_{th} place is the input place of the j_{th} transition. If the value in j_{th} row and n_{th} column of the incidence matrix is equal to 1, then n_{th} place is the output place of the j_{th} transition. While *Step 2.c.* runs for all places in any column of *P*, *Step 2* runs for all columns in P. Finally, in *Step 2.d* alternative route ***d*** is determined, and all alternative routes are listed in matrix *F*.

Assignments of tasks to the stations are done in *Step 3* using *F* matrix. Each column of *F* is an alternative route (or sequence) for this operation. In *Step 3.a.*, all columns of *F* are used to create assignment alternatives. In each cycle, same initial conditions must be used for each column (or for each sequence). Therefore alternatives having same token condition *MB*, same current station time *Swnow*, and same station number *ww* are created in the beginning of each cycle. During this algorithm *F* matrix is unchanged. Therefore selected task numbers remain in *F* after the assignment process. *Step 3.a.i.aa.* prevents a task from selecting over and over again. Once a task is selected, its number is placed in a list (*ASW*). If in any column of *F*, any selected task number is equal to any placed number in the list *ASW*, then the value of control variable a is changed from 0 to 1. In *Step 3.i.a.bb.*, nothing is done if the value of a is equal to 1. Therefore *Step 3.a.i.bb.* operates for unselected tasks.

Available task numbers are determined in *Step 3.a.i.bb.aaa.* Times of available tasks are compared with current station's idle time. In *Step 3.a.i.bb.bbb,* these candidates are temporarily allocated to the station until no allocation to the station is possible. Information for each temporary allocation is stored in *AL*, like as in *SN* vector in the previous algorithms. All *AL*s are listed in *ALS* matrix. For all columns of *F*, *Step 3.a.i.bb.bbb* is repeated. When all candidates are searched but no temporary allocation is created (*ALS*=[]) *Step 3.b.* is operated, that is, a new station is opened. If *ALS* is not empty, then number of candidates making idle time minimum is found using *ALS* matrix (*Step 3.c.*). Available tasks are allocated to the station, their numbers are added in the list of allocated tasks (*ASW*), and new token condition in the net is calculated in *Step 3.c.i.* In *Step 3.c.ii.*, number of allocated tasks (*awr*) is determined and *ALS* matrix is cleaned for new temporary allocations. All operations in *Step 3* are repeated until all tasks are allocated (*tn*>*awr*).

Step 4 is the step of displaying the results like as in the previous algorithms. Stations number, task number, task time, and idle time of the stations are given in *SON*. Line efficiency is also found and displayed.

4.5 Complexity Analysis of the Algorithms

In this section, computational complexities of the proposed algorithms are searched. Big-O notation is used to show the computational complexity of the algorithms (Nance & Naps, 1992, pp. 651-700).

4.5.1 Complexity Analysis for Reachability Analysis Algorithm

Step 2 in the algorithm of the reachability analysis algorithm is repeated at least *tn* times. Because each allocation process is done for one task, and therefore at least *tn* times allocation processes must be done. In each allocation process, all elements of the incidence matrix are investigated to determine the available transitions (*Step 2.a*). Therefore, for one allocation $tn \times pn$ times, and for all allocations $tn \times tn \times pn$ times investigations are made. In the selection step (*Step 2.b.*), *vr* times, which is the number of candidates in the current allocation, searches are made for one allocation, and therefore $tn \times vr$ times searches are made totally. Total operation number for this algorithm is at least $tn \times [(tn \times pn) + vr]$. Therefore complexity of the algorithm is shown in big-O notation such as $O(tn^2 \times pn)$. In other words, the computational complexity of the algorithm is related with the size of incidence matrix, square of row number (*or* square of transition number) and column number (or number of places) of the incidence matrix. Computation of complexity is again shown below.

$$O = tn \times [(tn \times pn) + vr]$$
$$O = tn^2 \times pn + tn \times vr \Rightarrow O(tn^2 \times pn)$$

For example, if Petri net model of a problem includes 14 places (or number of columns of the incidence matrix) and 10 transitions (or number of rows of the incidence matrix), then this algorithm needs totally 1400 (=10^2x14) operations to solve the problem.

4.5.2 Complexity Analysis for Firing Sequence of Transitions Algorithm

Sequencing operation (*Step 2)* is repeated at least *tn* times for this algorithm. Searches for availability of the transitions (*Step 2.a.i*) are made $tn \times pn$ times for one sequencing operation. In *Step 2.a.ii,* available transitions are compared with the transitions determined as available in the previous searches. This is necessary to keep information in *GT* matrix. This comparison is done *sl* times for one available transition, and $tn \times s1$ times for all available transitions. In *Step 2.b.*, one transition among candidates is placed in the sequencing list at the end of the *vr* times searches for one sequencing operation. For all sequencing operations, searching available transitions and comparison and selection operations are done $tn \times tn \times pn$, $tn \times tn \times s1$ and $tn \times vr$ times, respectively. Number of total assignment operations (*Step 3*) is *tn.* Therefore complexity of this algorithm is formalized as $O = tn \times [(tn \times pn) + (tn \times sl) + vr] + tn$, and represented with $O(tn^2 \times pn)$. Complexities of this algorithm and the previous algorithm are the same. That is, they are related with the size of the incidence matrix. Computation is given below.

$$O = tn \times [(tn \times pn) + (tn \times sl) + vr] + tn$$
$$O = tn^2 \times pn + tn^2 \times sl + tn \times vr + tn \Rightarrow O(tn^2 \times pn)$$
$$because \quad pn \geq sl$$

If the previous example is considered to compute the complexity of this algorithm then this algorithm, too, requires at least 1400 operations to solve the problem.

4.5.3 Complexity Analysis for Firing Sequence of Transitions-First Search Algorithm and Firing Sequence of Transitions-Best Search Algorithm

Complexity analyses of the third and fourth algorithms are similar. Therefore their complexities are shown in this subsection. First part of the complexity $(O = tn \times [(tn \times pn) + (tn \times sl) + vr])$ is the same as first part of the complexity of the previous algorithm (*Step 2*). Because firing sequences in these three algorithms (FS, FS-f, FS-b) are determined in the same way. For task allocation, Firing Sequence of Transitions-First Search Algorithm and Firing Sequence of Transitions-Best Search Algorithm need more searches than Firing Sequence of Transitions Algorithm. When the first task in the firing sequence of transitions is not allocated, available tasks are determined by searching all tasks in the *NSEQ*. Therefore at least $tn \times pn$ times searches are made for one allocation (*Step 3.b.i*). Also *cr* times operations are made for one allocation (*Step 3.b.ii*). In these algorithms, totally $tn \times tn \times pn$ and $tn \times vr$ operations are made for searching and allocation, respectively. When the firing sequencing is added to searching operations, complexity of the two algorithms is formulated as $O = tn \times [(tn \times pn) + (tn \times sl) + vr] + tn \times [(nsr \times pn) + cr]$. These two algorithms have the same complexity as the previous algorithms. That is, their complexities, too, are related with the size of incidence matrix. Computations are presented below.

$$O = tn \times [(tn \times pn) + (tn \times sl) + vr] + tn \times [(nsr \times pn) + cr]$$
$$O = tn^2 \times pn + tn^2 \times sl + tn \times vr + tn^2 \times pn + tn \Rightarrow O(tn^2 \times pn)$$
$$because \quad pn \geq sl \quad and \quad tn \geq nsr$$

These two algorithms need totally 1400 operations to solve the example problem.

4.5.4 Complexity Analysis for *P*-invariants Algorithm

In this algorithm, *P*-invariants are extra inputs when compared with the previous algorithms. In *Step 2*, alternative routes are determined using the *P*-invariants. All places in one *P*-invariant are searched to determine the place numbers in the subnet (*Step 2.a*). This step is repeated *pc* times for one *P*-invariant. Each transition is determined in *hr*-1 operations using its input and output places. (*Step 2.c.*), where *hr* is equal to number of places in any *P*-invariant. In worst case, *hr* is equal to *pn*. *Step 2.a* and *Step 2.c* are repeated for *pr* (number of *P*-invariants) times until the algorithm is completed. In *Step 3*, tasks are allocated using *F*, which includes alternative routes. Each column of *F* is one alternative route, and for each alternative route, *fr* (number of tasks in the route) times searches are made to determine the tasks to be allocated (*Step 3.a*). In worst case, *fr* is equal to *pn*. Also *awr* times tests are done for one task on the route to prevent a task from allocating repeatedly (*Step 3.a.i.aa*). *awr* is equal to *tn*-1 for worst case. This operation is done $fr \times awr$ times for one route, $fc \times fr \times awr$ times for all routes and for one task, and $tn \times fc \times fr \times awr$ times for all routes and for all tasks. According to the token condition, another search is made to determine the availability of the transition (task) by investigating the related row of the incidence matrix (*Step 3.a.i.bb*). This operation is repeated *pn* times for one task, $fr \times pn$ times for one route, $fc \times fr \times pn$ times for all routes and for one task, and $tn \times fc \times fr \times pn$ times for all tasks. In *Step 3.c.i.*, selected tasks (*alr*) are assigned to the station. This operation is iterated *alr* times for one allocation stage. Therefore, in worst case, *Step 3.c.i* is repeated $tn \times alr$ times until the algorithm is completed. All operations are formulized as $O = pr \times [pn + (pn-1) \times tn] + tn \times [pr \times tn \times ((tn-1) + pn) + alr]$, Complexity of the algorithm is related to number of transitions (or row numbers of the incidence matrix or number of tasks in the problem) and number of *P*-invariants. Computations are shown below.

If the Petri net model of the problem has four *P*-invariants, then totally 4000 ($=10^3$x4) operations are necessary to solve the problem using this algorithm.

$$O = pr\times[pc+(hr-1)\times tn]+tn\times[fc\times fr\times(awr+pn)+alr]$$
$$O = pr\times[pn+(pn-1)\times tn]+tn\times[pr\times tn\times((tn-1)+pn)+alr]$$
$$O = pr\times pn+pr\times pn\times tn-pr\times tn+tn\times pr\times tn\times tn-tn\times pr\times tn+tn\times pr\times tn\times pn+tn\times alr$$
$$O = pr\times pn+pr\times pn\times tn-pr\times tn+tn^3\times pr-tn^2\times pr+tn^2\times pr\times pn+tn\times alr \Rightarrow O(tn^3\times pr)$$
$$because\ \ pc=pn,\ \ hr\le pn,\ \ fc=pr,\ \ and\ \ awr\le tn-1$$

4.6 An Example

In this section, to explain the proposed algorithms, a simple assembly line balancing problem-type 1 is given and solved step by step. Precedence diagram of tasks with times is given in figure 4.6. Petri net model shown in figure 4.7 is built considering the precedence diagram.

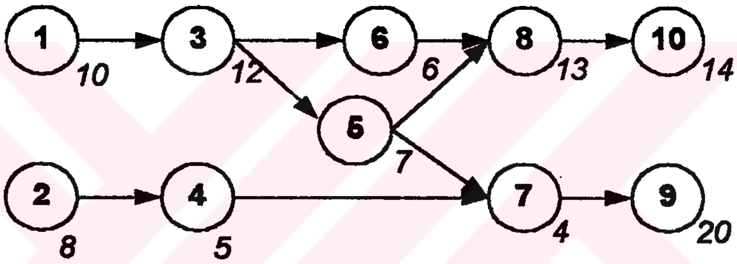


Figure 4.6 Precedence diagram of tasks

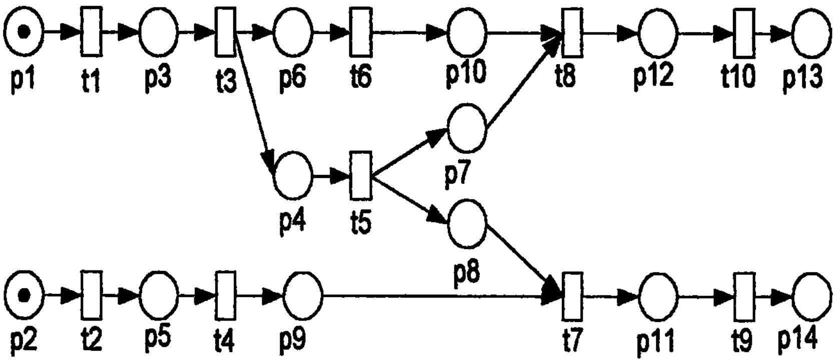


Figure 4.7 Petri net model of the precedence diagram

Two source places p_1 and p_2 in the net describes that in the precedence diagram, there are two tasks having no predecessor task. Two sink places p_{13} and p_{14} describes two tasks having no successor task. Furthermore, precedence diagram includes 10 relations, therefore Petri net model have 10 places having exactly one input and one output transition.

Inputs of the algorithms, which are incidence matrix, *P*-invariants, initial token condition, times of tasks, and cycle time, are given below.

$$
A=\begin{array}{c}
\begin{array}{cccccccccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} & p_{14} \end{array} \\
\left[\begin{array}{cccccccccccccc}
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0
\end{array}\right]
\end{array}
\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{array}
,\quad
P=\left[\begin{array}{cccc}
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1
\end{array}\right]
\begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_9 \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \end{array}
$$

$$
M=\begin{array}{c}
\begin{array}{cccccccccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} & p_{14} \end{array} \\
[\begin{array}{cccccccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}]
\end{array},
$$

$$
t=\begin{array}{c}
\begin{array}{cccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \end{array} \\
[\begin{array}{cccccccccc} 10 & 8 & 12 & 5 & 7 & 6 & 4 & 13 & 20 & 14 \end{array}]
\end{array},\quad C=35
$$

Columns of A represent the places, and rows represent the transitions. Since there are two source places, p_1 and p_2, having only one output transition, then sum of the elements in each of these columns must be equal to -1. There are also two sink places in the net, p_{13} and p_{14}, having only one input transition. Thus, sum of the elements in each of these columns must be equal to +1. Rows of the matrix represent the transitions, and the row elements have some meanings. For example, sum of the elements in the third row is equal to 1. It means that the difference between number of output places and number of input places of t_3 is one. In the beginning, M shows the initial number of tokens in places, and then it provides token movement until the end of the algorithm. Since there are one token in p_1 and p_2, the first two values of M are 1.

All steps of each solution algorithm are explained in the following subsections.

4.6.1 Solving with Reachability Analysis Algorithm

SALBP-1 problem is solved firstly using the reachability analysis algorithm which was explained in section 4.4.1. Inputs of the algorithm are incidence matrix, initial token condition, times of tasks, and cycle time. Table 4.1 shows all iterations of the solution.

In the beginning, initial token condition, station time, idle time, and values of these variables are given in the initial iteration. According to the initial token condition and incidence matrix, there are two firable transitions, t_1 and t_2. In other words, task 1 and task 2 are two alternatives for first assignment operation. Matrix *MY* includes first and second rows of incidence matrix. Matrix *O* shows the available tasks; task 1 and task 2 and their times; 10 and 8 time units (*Step 2.a.*). In the beginning, available time in station 1 for assignment process is equal to 35 time units (*C-Sw*=35-0). *Step 2.b.i* is skipped because there isn't any alternative task having 35 time units. *Step 2.b.ii* is run for alternatives, *Bz* values of task 1 and task 2 are calculated as 25 and 27 time units, respectively. Minimum *Bz*'s value is assigned to variable *minBz*, and task 1 is allocated to station 1. Vector *SN* is created for task 1. Since task 1 is assigned, vector *MN* used to determine the new token condition in the net, is created to select first row from matrix *MY*. In the second iteration, available tasks are task 2 and task 3 according to the new token condition *M*. *MY* is again formed by selecting second and third rows from *A*, and *O* is created. *Bz* values are calculated as 17 time units for task 2 and 13 time units for task 3 (*Step 2.b.ii.).* 13 time units is assigned to *minBz*, and task 2 to station 1, value of *Sw* is calculated as 22 time units, and as a result of assignment of task 2. *MN* is made equal to second row of *MY* (*Step 2.c.*).

Task 2, task 5, and task 6 arise as candidates from the new token condition. Therefore, matrix *MY* includes second, fifth, and sixth rows of incidence matrix. Elements of matrix *O* are the available task numbers; 2, 5, and 6 and their times; 8, 7, and 6 time units. *minBz* is determined as 5, 6, and 7 time units, respectively for these tasks, and task 2 given minimum *minBz* is selected for assignment. *MN* is created, and

Table 4.1 Iterations of Solution Using Reachability Analysis Algorithm

It.	*M*	*MY*	*O*	*w*	*Sw*	*Bz*	*minBz*	*SN*	*MN*	*SumOt*
In.	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]	[]	[]	1	0	35	35	[]	[]	0
1	[0 1 1 0 0 0 0 0 0 0 0 0 0 0]	$\begin{bmatrix} -1 & 0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0 & -1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}$	$\begin{bmatrix} 1 & 10 \\ 2 & 8 \end{bmatrix}$	1	0 10	25 for task 1 27 for task 2	25 25 25	[1 1 10 25]	[-1 0 1 0 0 0 0 0 0 0 0 0 0 0]	10
2	[0 1 0 1 0 1 0 0 0 0 0 0 0 0]	$\begin{bmatrix} 0\,-1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,-1\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}$	$\begin{bmatrix} 2 & 8 \\ 3 & 12 \end{bmatrix}$	1	22	17 for task 2 13 for task 3	17 13 13	[1 3 12 13]	[0 0 -1 1 0 1 0 0 0 0 0 0 0 0]	22
3	[0 0 0 1 1 1 0 0 0 0 0 0 0 0]	$\begin{bmatrix} 0\,-1\,0\;\;0\,1\;\;0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\;\;0\,0\,-1\,0\;\;0\,1\,1\,0\,0\,0\,0\,0\,0 \\ 0\;\;0\,0\;\;0\,0\,-1\,0\,0\,0\,1\,0\,0\,0\,0 \end{bmatrix}$	$\begin{bmatrix} 2 & 8 \\ 5 & 7 \\ 6 & 6 \end{bmatrix}$	1	30	5 for task 2 6 for task 5 7 for task 6	5 5 5 5	[1 2 10 5]	[0 -1 0 0 1 0 0 0 0 0 0 0 0 0]	30
4	[0 0 0 1 0 1 0 0 1 0 0 0 0 0]	$\begin{bmatrix} 0\,0\,0\;\;0\,-1\;\;0\,0\,0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,-1\;\;0\;\;0\,1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\;\;0\;\;0\,-1\,0\,0\,0\,1\,0\,0\,0\,0 \end{bmatrix}$	$\begin{bmatrix} 4 & 5 \\ 5 & 7 \\ 6 & 6 \end{bmatrix}$	1	35	0 for task 4	0 0	[1 4 5 0]	[0 0 0 0 -1 0 0 0 1 0 0 0 0 0]	35
5	[0 0 0 1 0 1 0 0 1 0 0 0 0 0]	$\begin{bmatrix} 0\,0\,0\,-1\,0\;\;0\,1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\;\;0\,0\,-1\,0\,0\,0\,1\,0\,0\,0\,0 \end{bmatrix}$	$\begin{bmatrix} 5 & 7 \\ 6 & 6 \end{bmatrix}$	1 2	0	-7 for task 5 -6 for task 6	35	[]	[]	35
6	[0 0 0 0 0 1 1 1 1 0 0 0 0 0]	$\begin{bmatrix} 0\,0\,0\,-1\,0\;\;0\,1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\;\;0\,0\,-1\,0\,0\,0\,1\,0\,0\,0\,0 \end{bmatrix}$	$\begin{bmatrix} 5 & 7 \\ 6 & 6 \end{bmatrix}$	2	7	28 for task 5 29 for task 6	28 28 28	[2 5 7 28]	[0 0 0 -1 0 0 1 1 0 0 0 0 0 0]	42

Table 4.1 (Cont.)

It.	*M*	*MY*	*O*	*w*	*Sw*	*Bz*	*minBz*	*SN*	*MN*	*SumOt*
7	[0 0 0 0 0 0 1 1 1 1 0 0 0 0]	$\begin{bmatrix} 0&0&0&0&0&-1&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&-1&-1&0&1&0&0&0 \end{bmatrix}$	$\begin{bmatrix} 6&6 \\ 7&4 \end{bmatrix}$	2	13	22 for task 6 24 for task 7	22 22 22	[2 6 6 22]	[0 0 0 0 0 -1 0 0 0 1 0 0 0 0]	48
8	[0 0 0 0 0 0 0 1 1 0 0 1 0 0]	$\begin{bmatrix} 0&0&0&0&0&0&0&-1&-1&0&1&0&0&0 \\ 0&0&0&0&0&0&-1&0&0&-1&0&1&0&0 \end{bmatrix}$	$\begin{bmatrix} 7&4 \\ 8&13 \end{bmatrix}$	2	26	15 for task 7 9 for task 8	15 9 9	[2 8 13 9]	[0 0 0 0 0 0 -1 0 0 -1 0 1 0 0]	61
9	[0 0 0 0 0 0 0 0 0 0 1 1 0 0]	$\begin{bmatrix} 0&0&0&0&0&0&0&-1&-1&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&-1&1&0 \end{bmatrix}$	$\begin{bmatrix} 7&4 \\ 10&14 \end{bmatrix}$	2	30	5 for task 7 -5 for task 10	5 5	[2 7 4 5]	[0 0 0 0 0 0 0 -1 -1 0 1 0 0 0]	65
10	[0 0 0 0 0 0 0 0 0 0 1 1 0 0]	$\begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&-1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&-1&1&0 \end{bmatrix}$	$\begin{bmatrix} 9&20 \\ 10&14 \end{bmatrix}$	2 3	0	-15 for task 9 -9 for task 10	35	[]	[]	65
11	[0 0 0 0 0 0 0 0 0 0 0 1 0 1]	$\begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&-1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&-1&1&0 \end{bmatrix}$	$\begin{bmatrix} 9&20 \\ 10&14 \end{bmatrix}$	3	20	15 for task 9 21 for task 10	15 15 15	[3 9 20 15]	[0 0 0 0 0 0 0 0 0 0 -1 0 0 1]	85
12	[0 0 0 0 0 0 0 0 0 0 0 0 1 1]	$\begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&-1&1&0 \end{bmatrix}$	[10 14]	3	34	1 for task 10	1 1	[3 10 14 1]	[0 0 0 0 0 0 0 0 0 0 0 -1 1 0]	99

current *Sw* is calculated as 30 time units. In the fourth iteration, task 4, task 5, and task 6 are found as candidates. Time of task 4 provides the zero idle time for station 1. Therefore task 4 is assigned to station 1 (*Step 2.b.i.*). *Step 2.b.i.* or *Step 2.b.ii.* isn't operated for tasks 5 and 6. Values of *minBz*, and *Sw* are 0 and 35 time units, respectively. Vector *MN* is made equal to first row of *MY*. According to the new token condition, tasks 5 and 6 are available. However *Bz* values corresponding to these tasks are negative, because their times are bigger than the *Sw*, therefore *Step 2.b.i.* and *Step 2.b.ii.* are skipped. Since there is no assignment in this stage (*Oz*=[]), *Step 2.c.i.* is taken into account. New station is opened. Its idle time (*minBz*) is equal to 35 time units (*C*), and its task time (*Sw*) is equal to 0 time units. Token condition in the net doesn't change, because *MN* is a zero vector. Task 5 and task 6 are used as candidates in the sixth iteration. *Bz* values are 28 and 29 time units for task 5 and 6, respectively. Thus 28 time units is determined as *minBz* and task 5 is selected to be assigned to second station. For the seventh iteration, new token condition is found using first row of *MY*. Task 6 and task 7 are the new alternatives, and their times, 6 and 4 time units, are stored in the second column of *O*. Values of *Bz* for task 6 and task 7 are 22 and 24 time units, respectively. Minimum *Bz* value (22 time units) is determined as *minBz*, and task 6 is selected for assignment. Elements of vector *SN* are formed for task 6, and *MN* is created to determine the new token condition. In the eighth iteration, task 8 is selected from alternative tasks (task 7 and task 8), because its *Bz* value (9 time units) is less than time (15 time units) of task 7. *SN* and *MN* are created for task 8. According to the new token condition, task 7 and task 10 are alternatives. Since *Bz* value of task 10 is negative (-5), task 7 is selected to assign, and any operation isn't done for task 10. *Bz* value (5) of task 7 is determined as *minBz*. *SN* and *MN* are created according to the assigned task 7. In the 10_{th} iteration, task 9 and task 10 are determined as alternatives, but any assignment couldn't be done, since their *Bz* values are negative. Therefore *Step 2.b.i.* and *Step 2.b.ii.* are skipped. Thus new station is opened to make any assignment from alternatives. Values of *minBz*, and *Sw* are updated as their initial values. Task 9 and task 10 are alternatives in iteration 11, too. *Bz* values of task 9 and task 10 are 15 and 21 time units, respectively. Therefore, 15 is determined as *minBz*, and then task 9 is assigned to third station. In the last iteration, there is only one alternative, task 10. Thus, task 10 is allocated to station 3. At the end of the each assignment in *Step 2*, total time of assigned tasks (*SumOt*) is calculated and compared with total of task times (*Sumt*). If *SumOt* is equal to *Sumt*, algorithm is finished. After the all assignments, total time of all tasks is equal to 99 in the end of the 12_{th} iteration. *SON* matrix includes the *SN* vectors created in the end of the all iterations for assigned tasks (*Step 2.c*). Finally line efficiency is calculated. *SON* matrix and line efficiency is given below.

w	tn	t	Bz
1	1	10	25
1	3	12	13
1	2	10	5
1	4	5	0
2	5	7	28
2	6	6	22
2	8	13	9
2	7	4	5
3	9	20	15
3	10	14	1

Line efficiency of assembly (LE) = 0.9429

4.6.2 Solving with Firing Sequence of Transitions Algorithm

This algorithm contains two main steps. In the first step, firing sequence is found by a dynamic study and using times of tasks (or transitions). In table 4.2, all iterations of solutions are given.

This algorithm uses same inputs as reachability analysis algorithm. In initial, token condition (*M*) represents the initial token condition. *O*, *MN*, and *SEQ* are empty matrix, and values of *time*, *Oz* and *sr* are equal to 0. In *Step 2.a.* first and second transitions are determined as enabled transitions considering the initial token condition, and *O* matrix is created. It includes transitions' numbers; 1 and 2 and firing times 10 and 8 time units. In *Step 2.b.*, second transition, which has minimum firing completion time is preferred to first, 2 is added to sequencing list (*SEQ*), value of *time* is made equal to 8, *MN* is updated using second row of incidence matrix to find new token condition, value of number of sequenced transitions (*sr*) (is made equal to 1, and information related with transition 2 is deleted from *O* (*Step 2.b.ii.*). So first iteration is finished. In the second iteration, according to the new token condition transition 4 can fire. Also transition 1 has started to fire but hasn't finished yet during the value of

Table 4.2 Determination of Firing Sequence of Transitions

It.	*M*	*O*	*time*	*Oz*	*MN*	*SEQ'*	*sr*
In.	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]	[]	0	0	[]	[]	0
1	[1 0 0 0 1 0 0 0 0 0 0 0 0 0]	$\begin{bmatrix} 1 & 10 \\ 2 & 8 \end{bmatrix}$	0 8	0 2	[0 -1 0 0 1 0 0 0 0 0 0 0 0 0]	[2]	1
2	[0 0 1 0 1 0 0 0 0 0 0 0 0 0]	$\begin{bmatrix} 1 & 10 \\ 4 & 13 \end{bmatrix}$	10	1	[-1 0 1 0 0 0 0 0 0 0 0 0 0 0]	[2 1]	2
3	[0 0 1 0 0 0 0 0 1 0 0 0 0 0]	$\begin{bmatrix} 4 & 13 \\ 3 & 22 \end{bmatrix}$	13	4	[0 0 0 0 -1 0 0 0 1 0 0 0 0 0]	[2 1 4]	3
4	[0 0 0 1 0 1 0 0 1 0 0 0 0 0]	[3 22]	22	3	[0 0 -1 1 0 1 0 0 0 0 0 0 0 0]	[2 1 4 3]	4
5	[0 0 0 1 0 0 0 0 1 1 0 0 0 0]	$\begin{bmatrix} 5 & 29 \\ 6 & 28 \end{bmatrix}$	28	6	[0 0 0 0 0 -1 0 0 0 1 0 0 0 0]	[2 1 4 3 6]	5
6	[0 0 0 0 0 0 1 1 1 0 0 0 0 0]	[5 29]	29	5	[0 0 0 -1 0 0 1 1 0 0 0 0 0 0]	[2 1 4 3 6 5]	6
7	[0 0 0 0 0 0 1 0 0 1 1 0 0 0]	$\begin{bmatrix} 7 & 33 \\ 8 & 42 \end{bmatrix}$	33	7	[0 0 0 0 0 0 0 -1 -1 0 1 0 0 0]	[2 1 4 3 6 5 7]	7
8	[0 0 0 0 0 0 0 0 0 0 1 1 0 0]	$\begin{bmatrix} 8 & 42 \\ 9 & 53 \end{bmatrix}$	42	8	[0 0 0 0 0 0 -1 0 0 -1 0 1 0 0]	[2 1 4 3 6 5 7 8]	8
9	[0 0 0 0 0 0 0 0 0 0 0 1 0 1]	$\begin{bmatrix} 9 & 53 \\ 10 & 56 \end{bmatrix}$	53	9	[0 0 0 0 0 0 0 0 0 0 -1 0 0 1]	[2 1 4 3 6 5 7 8 9]	9
10	[0 0 0 0 0 0 0 0 0 0 0 0 1 1]	[10 56]	56	10	[0 0 0 0 0 0 0 0 0 0 0 -1 1 0]	[2 1 4 3 6 5 7 8 9 10]	10

time equals to 8. Therefore, time point at which it stopped to firing is kept in *O* (*Step 2.a.ii.*). Also firing completion time of transition 4 (*i.e.* 13 time units) which is equal to its firing time (5 time units) plus the value of *time* (8 time units) is added in *O*. Since transition 1 has minimum firing completion time (i.e. 10), it is added in sequencing list in iteration 2, and value of *time* is made equal to 10. Same algorithms are repeated until all transitions are sequenced. In iteration 3, transition 4, which was placed in iteration 2, and transition 3, which can fire according to the new token condition, are candidates to add in sequencing list. Firing completion time of transition 4, which was calculated in the second iteration, is 13 time units. Firing completion time of transition 3 (i.e. 22 time units) is found by adding its firing time (12) to the value of *time* (10). 13 is the minimum firing completion time, thus 4 is placed into sequencing list, and value of *time* is made equal to 13 for new sequencing. In iteration 4, new token condition doesn't give any new candidate transition to be fired. Therefore transition 3 is added to the list (*SEQ*), and *time* is determined as 22. In iteration 5, there are two candidates, and their firing completion times are 29 for transition 5 and 28 for transition 6. According to their firing completion times, transition 6, which has minimum firing completion time, is placed into the sequencing list and *time* is determined as 28. There isn't any addition of transition number into *O*, because new token condition prevents any transition firing except transition 5. Thus transition 5 is added to the list and *time* is made equal to 29. New token condition in the next iteration satisfies firing transition 7 and 8. Their firing completion times are 33 for transition 7 and 42 for transition 8. Transition 7 is selected and added in the list and *time* is 33 now. In iteration 8, transition 8 is selected from two candidates, transition 8 and transition 9, and *time* is determined as 42. New token condition in iteration 9 determines transition 10 as a new candidate. Also transition 9 is again other candidate. Transition 9 is selected and added in the list and *time* is made equal to 53. In the last iteration, there is only one transition waiting for firing, transition 10. It is placed into list, and *Step 2* in the algorithm is completed. A firing sequencing is created to use in the next steps for assignments, such as 2, 1, 4, 3, 6, 5, 7, 8, 9, and 10.

Second main step of the algorithm is the allocation tasks to stations according to the firing sequencing. All iterations of this allocation stage are given in table 4.3. In the initial condition, token condition is made equal to initial token condition, idle time is made equal to cycle time (*Bz*=35), station number is 1, *SN* is empty, and total of times of allocated tasks is 0 (*SumOt*). Also *NSEQ* is determined as the firing sequencing (*SEQ*). At the end of the each assignment operation, *NSEQ* is continuously updated by deleting allocated task number from *NSEQ*.

In iteration 1, first task in sequencing list (2) is allocated to station 1 using *Step 3.b*. Because the current idle time (*Bz*=35) is bigger than to time of task 2 (8 time units). *SN* is created, and it includes information of station number, task number, time of task, and idle time. New sequencing list (*NSEQ*) is updated by deleting task 2 from the list. In the second iteration, task 1 is selected into the list, and allocated to station 1. Because of that idle time of station 1 (*Bz*=27) is bigger than time of task 1 (10 time units). *SN* and new sequencing list are created in similarly as in previous iteration. Task 4 and task 3 are assigned to station 1 in iteration 3 and iteration 4, respectively using *Step 2.b*. After the iteration 4, *Bz* is equal to 0, therefore *Step 3.a.* works to open

Table 4.3 Iterations of Solution Using Firing Sequencing Algorithm

It.	*M*	*NSEQ'*	*S.J.**	*J.T.***	*Bz*	*w*	*SN*	*SumOt*
In.	[1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]	[2 1 4 3 6 5 7 8 9 10]	0	0	35	1	[]	0
1	[1 0 0 0 1 0 0 0 0 0 0 0 0 0 0]	[1 4 3 6 5 7 8 9 10]	2	8	27	1	[1 2 8 27]	8
2	[0 0 1 0 1 0 0 0 0 0 0 0 0 0 0]	[4 3 6 5 7 8 9 10]	1	10	17	1	[1 1 10 17]	18
3	[0 0 1 0 0 0 0 0 1 0 0 0 0 0 0]	[3 6 5 7 8 9 10]	4	5	12	1	[1 4 5 12]	23
4	[0 0 0 1 0 1 0 0 1 0 0 0 0 0 0]	[6 5 7 8 9 10]	3	12	0	1	[1 3 12 0]	35
5	[0 0 0 1 0 1 0 0 1 0 0 0 0 0 0] [0 0 0 1 0 0 0 0 1 1 0 0 0 0 0]	[6 5 7 8 9 10] [5 7 8 9 10]	6	6	0 35 29	1 2	[2 6 6 29]	41
6	[0 0 0 0 0 0 1 1 1 0 0 0 0 0 0]	[7 8 9 10]	5	7	22	2	[2 5 7 22]	48
7	[0 0 0 0 0 0 1 0 0 1 1 0 0 0 0]	[8 9 10]	7	4	18	2	[2 7 4 18]	52
8	[0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]	[9 10]	8	13	5	2	[2 8 13 5]	65
9	[0 0 0 0 0 0 0 0 0 0 0 0 1 0 1] [0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]	[9 10] [10]	9	20	35 15	2 3	[] [3 9 20 15]	65 85
10	[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]	[]	10	14	1	3	[3 10 14 1]	99

* S.J. :Selected and allocated task number from firing sequencing list (NSEQ)
** J.T.: Time of selected and allocated task

second station and its idle time is made equal to cycle time (*w*=2, *Bz*=35). Task 6 in the sequence list is available for assignment to second station (*Step 2.b.*). After this assignment, idle time of second station decreases to 29 time units. Task 5, task 7, and task 8 are allocated to station 2 in iterations 6, 7,and 8, respectively. After the allocation of task 8, idle time of station 2 is determined as 5 time units. In the next iteration, task 9 is selected for allocation according to the sequencing list (*NSEQ*). But task 9 is not allocated to the station 2, because its time (20 time units) is bigger than the current idle time (*Bz*=5). In this condition, *Step 2.a.* works in algorithm to open new station (*w*=3) and to determine its idle time as 35. Task 9 and task 10 is allocated to station 3 in iterations 9 and 10, respectively. When the total time of allocated tasks (*SumOt*) is equal to total time of tasks (*Sumt*), *Step 3* is finished.

In *Step 4*, all *SN* vectors, which are created in the end of the each allocation algorithm, are displayed in a matrix (*SON*). Also line efficiency is determined according to the allocations. Results are given as,

w	tn	t	Bz
1	2	8	27
1	1	10	17
1	4	5	12
1	3	12	0
2	6	6	29
2	5	7	22
2	7	4	18
2	8	13	5
3	9	20	15
3	10	14	1

Line efficiency of assembly (LE) = 0.9429

4.6.3 Solving with Firing Sequence of Transitions-First Search Algorithm

This algorithm is similar with previous algorithm therefore all of the solving iterations don't be explained in details. All operations in the previous algorithm for determining the firing sequencing are similar with the operations in this algorithm. Therefore all iterations in table 4.2 and explanations of these iterations are valid for this algorithm. Also allocation operations till assignment of the task 8 to second station are same with previous algorithm. Therefore first 8 iterations in table 4.3 and their explanations are valid for this algorithm, too. After the allocation task 8, the algorithm works differently. Last 3 iterations of the solving the example problem using the algorithm are given in table 4.4.

Bz is made equal to 5 time units and *w* is equal to 2, after the allocation of task 8. In iteration 9, time of task 9 (20 time units) is compared with *Bz* (5 time units). Time of task 9 is bigger than the *Bz*, thus next tasks in the sequence are searched and first available task is assigned to the station 2 (*Step 3.b.*). According to the token condition, availability of the tasks in the sequence is searched (*Step 3.b.i.*). Candidates are determined as task 9 and task 10 according to the token condition. Time of neither task 9 nor task 10 is less than the *Bz* (*Step 3.b.ii.*). Therefore any assignment is not

Table 4.4 Last 3 Iterations of Solution Using Firing Sequencing-First Search Algorithm

It.	*M*	*NSEQ'*	*CJ*	*S.J.**	*J.T.***	*Bz*	*w*	*SN*	*SumOt*
8				8	13		2		
	[0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]	[9 10]				5		[2 8 13 5]	65
9			$\begin{bmatrix} 9 & 20 \\ 10 & 14 \end{bmatrix}$				2	[]	
	[0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]	[9 10]				35	3		65
				9	20				
	[0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]	[10]				15		[3 9 20 15]	85
10				10	14		3		
	[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]	[]				1		[3 10 14 1]	99

* S.J. :Selected and allocated task number from firing sequencing list (*NSEQ*)
** J.T.: Time of selected and allocated task

made to the second station (a=0). Third station is opened and *Bz* is determined as 35 time units (*Step 3.b.iii.*), task 9 and task 10 are assigned to the new station.

Results are similar with results found from previous algorithm.

w	tn	t	Bz
1	2	8	27
1	1	10	17
1	4	5	12
1	3	12	0
2	6	6	29
2	5	7	22
2	7	4	18
2	8	13	5
3	9	20	15
3	10	14	1

Line efficiency of assembly (LE) = 0.9429

4.6.4 Solving with Firing Sequence of Transitions-Best Search Algorithm

This algorithm has same logic for determination of firing sequences and same assignments' results until last two tasks with previous two algorithms. Therefore operations in section 4.6.2. for determination of firing sequence and allocations of first 8 tasks are valid for this algorithm. Also last 3 iterations in the previous algorithm are similar with last three iterations in this algorithm. After the allocation of task 8, idle time (*Bz*) of second station is equal to 5 time units. Time of the first task in the sequence is bigger than the *Bz*. *CJ* are created in *Step 3.b.i.* same as in previous algorithm. In *Step 3.b.ii.*, a searching is made to determine the task given the minimum idle time into candidates. However candidates, task 9 and task 10, are not available for any allocation to second station, because their times are bigger than the *Bz*. Therefore any assignment is not made, and third station is opened. Allocation starts from task 9, which is the first task in the sequence. Finally task 10 is allocated the station and algorithm is finished. Last three iterations are same in table 4.4, and results of the algorithm are similar with results of the previous algorithm.

4.6.5 Solving with *P*-invariants Algorithm

It uses *P*-invariants to create alternative sequences. Inputs are the incidence matrix, *P*-invariants, initial token condition, times of tasks, and cycle time. Creating alternative routes using *P*-invariants, and allocating tasks to stations using these routes are two main steps of the algorithm. Table 4.5 shows how alternative routes are created for the problem considered.

Table 4.5 Determination of Alternative Routes

It.	*pr*	*P'(pr)*	*h*	*d*
1	1	[0 1 0 0 1 0 0 0 1 0 1 0 0 1]	[2 5 9 11 14]	[2 4 7 9]
2	2	[1 0 1 0 0 1 0 0 0 1 0 1 1 0]	[1 3 6 10 12 13]	[1 3 6 8 10]
3	3	[1 0 1 1 0 0 1 0 0 0 0 1 1 0]	[1 3 4 7 12 13]	[1 3 5 8 10]
4	4	[1 0 1 1 0 0 0 1 0 0 1 0 1 0]	[1 3 4 8 11 13]	[1 3 5 7 9]

Incidence matrix and *P*-invariants are used in the stage of determining *F* (*Step 2*). In the first iteration, in the first row of *P'*, if an element is strictly positive, its column number is listed in *h* vector (*Step 2.a.*). The elements of *h* (2,5,9,11,14) are used to specify transitions' numbers between the places on the route (*Step 2.c.*). Place numbers are searched for each transition (*Step 2.c.i.*). In this example, p_2 is determined as input place of t_2 using incidence matrix. That is, if value in second row and second column of incidence matrix is equal to –1, then this place is the input place of t_2. If the second value in h is equal to output place of t_2, then transition number (2) will be added in d (*Step 2.c.i.aa.*). Second number in the first row of P' is equal to 5. p_5 is output place of t_2, because value in the second row and fifth column of incidence matrix is 1. Therefore 2 is added in d. This algorithm (*Step 2.a.*) is repeated until all places in the rows of P' are searched. p_5 is the input place of t_4 and p_9 is the output place of t_4. Therefore 4 is added to d. p_9 and p_{11} is specified as input and output places of t_7. Now 7 is added in d. Finally, input place of t_9 is p_{11} and output place of t_9 is p_{13}. 9 is the final value added in d. At the end of the first iteration, elements of first alternative route are 2, 4, 7, 9. In the second iteration, elements of h are specified as 1,3,6,10,12,13 using the second row of P'. 1, 3, 6, 8, and 10 are transitions' numbers obtained using h and incidence matrix. Same algorithm is operated, and elements of h are found as 1, 3, 4, 7, 12, 13, and 1, 3, 4, 8, 11, 13, for third and fourth iterations, respectively. At the end of iteration 3, d=[1 3 5 8 10], and at the end of iteration 4, d=[1 3 5 7 9]. At the end of the each iteration, d is added to F, which includes all generating alternative routes, and d and h are empty vectors for new iterations. Thus, F is found as below. Zero value is added in the end of first column for displaying F in matrix format.

$$F = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 4 & 3 & 3 & 3 \\ 7 & 6 & 5 & 5 \\ 9 & 8 & 8 & 7 \\ 0 & 10 & 10 & 9 \end{bmatrix}$$

Second main step is the allocation of tasks using F. Each column of F is an alternative sequence (subset of transitions). All iterations in allocation stage are given in table 4.6. In the first iteration, variables used in allocation stage take initial values. Token condition is made equal to initial token condition (*M=MB*), current station time is equal to 0 (*Swnow=Sw*), and current station number is equal to 1 (*ww=w*). This is done in each running of *Step 3.a.i.* Therefore same initial conditions are provided to analyze each alternative sequence. Also *AL* and *ALS*, which keep the required allocation information, are empty in initial iteration. In the first iteration, all

Table 4.6 Iterations of Allocation Process Using *F*

It.	*M*	*fc*	*JN*	*JT*	*Swnow*	*Sw*	*ww*	*w*	*AL*	*ALS*	*SJ*	*awr*
In.	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]				0	0	1	1	[]	[]		0
1		1	2	8	8		1		[1 1 2 8 8 27]			
	[1 0 0 0 1 0 0 0 0 0 0 0 0 0]											
		1	4	5	13		1		[1 1 4 5 13 22]			
	[1 0 0 0 0 0 0 0 1 0 0 0 0 0]											
	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]				0		1					
		2	1	10	10		1		[2 1 1 10 10 25]			
	[0 1 1 0 0 0 0 0 0 0 0 0 0 0]											
		2	3	12	22		1		[2 1 3 12 22 13]			
	[0 1 0 1 0 1 0 0 0 0 0 0 0 0]											
		2	6	6	28		1		[2 1 6 6 28 7]			
	[0 1 0 1 0 0 0 0 0 1 0 0 0 0]											
	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]				0		1					
		3	1	10	10		1		[3 1 1 10 10 25]			
	[0 1 1 0 0 0 0 0 0 0 0 0 0 0]											
		3	3	12	22		1		[3 1 3 12 22 13]			
	[0 1 0 1 0 1 0 0 0 0 0 0 0 0]											
		3	5	7	29		1		[3 1 5 7 29 6]			
	[0 1 0 0 0 1 1 1 0 0 0 0 0 0]											
	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]				0		1					
		4	1	10	10		1		[4 1 1 10 10 25]			
	[0 1 1 0 0 0 0 0 0 0 0 0 0 0]											
		4	3	12	22		1		[4 1 3 12 22 13]			
	[0 1 0 1 0 1 0 0 0 0 0 0 0 0]											
		4	5	7	29		1		[4 1 5 7 29 6]			
	[0 1 0 0 0 1 1 1 0 0 0 0 0 0]											
	[1 1 0 0 0 0 0 0 0 0 0 0 0 0]											
	[0 1 0 0 0 1 1 1 0 0 0 0 0 0]					29		1		$\begin{bmatrix} 1 & 1 & 2 & 8 & 8 & 27 \\ 1 & 1 & 4 & 5 & 13 & 22 \\ 2 & 1 & 1 & 10 & 10 & 25 \\ 2 & 1 & 3 & 12 & 22 & 13 \\ 2 & 1 & 6 & 6 & 28 & 7 \\ 3 & 1 & 1 & 10 & 10 & 25 \\ 3 & 1 & 3 & 12 & 22 & 13 \\ 3 & 1 & 5 & 7 & 29 & 6 \\ 4 & 1 & 1 & 10 & 10 & 25 \\ 4 & 1 & 3 & 12 & 22 & 13 \\ 4 & 1 & 5 & 7 & 29 & 6 \end{bmatrix}$	1,3,5	3
2		2	1	6	35		1		[2 1 6 6 35 0]			
	[0 1 0 0 0 0 1 1 0 1 0 0 0 0]											
										[2 1 6 6 35 0]		
						35		1			6	4
3										[]		4
	[0 1 0 0 0 0 1 1 0 1 0 0 0 0]					0		2				

JN: Selected task number
JT: Time of selected task

Table 4.6 (Cont.)

It.	*M*	*fc*	*JN*	*JT*	*Swnow*	*Sw*	*ww*	*w*	*AL*	*ALS*	*SJ*	*awr*
4		1	2	8	8		2		[1 2 2 8 8 27]			
	[0 0 0 0 1 0 1 1 0 1 0 0 0 0]											
		1	4	5	13		2		[1 2 4 5 13 22]			
	[0 0 0 0 0 0 1 1 1 1 0 0 0 0]											
		1	7	4	17		2		[1 2 7 4 17 18]			
	[0 0 0 0 0 0 1 0 0 1 1 0 0 0]											
	[0 1 0 0 0 0 1 1 0 1 0 0 0 0]				0		2					
		2	8	13	13		2		[2 2 8 13 13 22]			
	[0 1 0 0 0 0 0 1 0 0 0 1 0 0]											
		2	10	14	27		2		[2 2 10 14 27 8]			
	[0 1 0 0 0 0 0 1 0 0 0 0 1 0]											
	[0 1 0 0 0 0 1 1 0 1 0 0 0 0]				0		2					
		3	8	13	13		2		[3 2 8 13 13 22]			
	[0 1 0 0 0 0 0 1 0 0 0 1 0 0]											
		3	10	14	27		2		[3 2 10 14 27 8]			
	[0 1 0 0 0 0 0 1 0 0 0 0 1 0]											
	[0 1 0 0 0 0 1 1 0 1 0 0 0 0]											
										$\begin{bmatrix} 1 & 2 & 2 & 8 & 8 & 27 \\ 1 & 2 & 4 & 5 & 13 & 22 \\ 1 & 2 & 7 & 4 & 17 & 18 \\ 2 & 2 & 8 & 13 & 13 & 22 \\ 2 & 2 & 10 & 14 & 27 & 8 \\ 3 & 2 & 8 & 13 & 13 & 22 \\ 3 & 2 & 10 & 14 & 27 & 8 \end{bmatrix}$		
						27		2			8,10	6
	[0 1 0 0 0 0 0 1 0 0 0 0 1 0]											
5		1	2	8	35		2		[1 2 2 8 35 0]			
	[0 0 0 0 1 0 0 1 0 0 0 0 1 0]											
										[1 2 2 8 35 0]		
						35		2			2	7
6										[]		7
	[0 0 0 0 1 0 0 1 0 0 0 0 1 0]					0		3				
7		1	4	5	5		3		[1 3 4 5 5 30]			
	[0 0 0 0 0 0 0 1 1 0 0 0 1 0]											
		1	7	4	9		3		[1 3 7 4 9 26]			
	[0 0 0 0 0 0 0 0 0 0 1 0 1 0]											
		1	9	20	29		3		[1 3 9 20 29 6]			
	[0 0 0 0 0 0 0 0 0 0 0 0 1 1]											
										$\begin{bmatrix} 1 & 3 & 4 & 5 & 5 & 30 \\ 1 & 3 & 7 & 4 & 9 & 26 \\ 1 & 3 & 9 & 20 & 29 & 6 \end{bmatrix}$		
						29		3			4,7,9	10

alternative sequences are searched in order. First column of *F* is the first alternative sequence. According to the token condition in the net, t_2 can fire (*Step 3.a.i.bb.aaa.*) and task 2 is selected and allocated to station 1 (*Step 3.a.i.bb.bbb.*), because station's idle time (*C-Swnow*=35 time units) is greater than time of task 2 (8 time units). New *Swnow* is calculated by adding time of task 2 to *Swnow* (*i.e.* 8+0=8). Vector *AL* for task 2 is created using allocation information, such that value in first column of *AL* is the alternative number (*fc=1*), value in second column of *AL* is the station number (*ww*=1), value in third column of *AL* is the task number (*JN*=2), value in fourth column of *AL* is the time of task (*JT*=8), value in fifth column of *AL* is the station time (*Swnow*=8), and value in the last column of *AL* is idle time (*C-Swnow*=35-8=27). *AL* is added in *ALS* matrix which is used in assignment stage. New token condition is found by adding second row of incidence matrix to token condition vector *M*.

All these operations are done using *Step 2.a.i.bb.bbb.aaaa*. New token condition allows firing t_4, which is the second row of first column of *F*. Task 4 is allocated to station 1, because the current idle time (35-8=27) is bigger than the time of task 4 (*i.e.* 5). *AL* is created as in the previous task allocation stage, and added in *ALS*. New token condition is determined using last token condition and fourth row of incidence matrix. Next value in the first alternative sequence is 7, which represents the task 7. But task 7 couldn't be allocated, because it is not enabled. That is, among the input places of t_7 only p_9 has a token. Therefore t_7 can't fire, and algorithm backs to *Step 3.a*. For the second alternative sequence (*fc*=2), station time and token condition are determined as in the beginning of the first iteration. First value in the second column of *F* is 1, therefore task 1 is selected and allocated according to the token condition. *Swnow* is equal to 10, and *AL* is created for task 1 and added in *ALS*. Also new token condition is calculated by adding first row of incidence matrix to last token condition. For new allocation to station 1, considering the new token condition task 3 is selected, which is the second task in the second column of *F*. Time of task 3 (*i.e.* 12) is less than thecurrent idle time of station 1 (*i.e.* 25). Therefore task 3 is allocated to station 1 as the second task of this station. *Swnow* is calculated as 22 time units, *AL* is formed and placed into *ALS*. New token condition is determined using third row of incidence matrix. Third value in the second column of *F* is 6, and since t_6 is enabled and current idle time is greater than the time of task 6, task 6 can be selected for new allocation (*Swnow*=28). New token condition is found using sixth row of the incidence matrix. *AL* is created and added in *ALS*. Although the next value in the list represents task 8, task 8 is not allocated because it is not enabled. Thus *Swnow* and token condition are updated for new alternative sequence (*fc*=3). First two values in the third column of *F* are 1 and 3, and they represent the task 1 and task 3. These tasks are allocated as in the previous allocation stage. After these allocations, *Swnow* is equal to 22, and new token condition allows selecting task 5 whose number is the third value in the third column of *F*. Task 5 is the third task allocated to station 1 because its time (7) less than the current idle time of station 1 (35-22=13). *AL* is created and added in *ALS*. New selected task is task 8 according to the third alternative sequence. But current token condition is not appropriate for its allocation to station 1. Therefore *Swnow* and token condition are updated for last alternative sequence (*fc*=4). First three tasks in the last column of *F* and the first three tasks in the third column of *F* are the same. Therefore same operations are done, again. After the third allocation process, next task in *d* is task 7. But task 7 is not allocated to station 1 because it is not enabled. Thus all alternative routes are used for allocation under the same initial conditions. All results are given in *ALS*. Last column of *ALS* gives the idle time of station for each allocation process. Minimum value in the last column of *ALS* is equal to 6, which is found using third and fourth alternatives (*Step 3.c.*). Third alternative is the best, because the minimum idle time is firstly calculated by using third alternative sequence. *Sw* is equal to *Swnow*, *w* is equal to *ww*, and new token condition is equal to the token condition which is calculated after firing t_5. They are new initial conditions for the next allocation stages. In *Step 3.c.i.aa.*, allocated task numbers (*i.e.* 1,3,5) are collected from third column of *ALS*. Then, these numbers are placed into *ASW* which will be used by the next allocation stages, and at the end of the algorithm *SON* matrix is created to show the results for each allocated task (*Step 3.c.i.aa.*). In *Step 3.c.ii.*, number of allocated task (*awr*=3) is determined and *ALS* is cleaned for new allocations. Iteration 2 starts under the new token condition, which is created at the end of the firing t_1, t_3, and t_5. According to the token condition and idle time of station 1, only third value in the second column of *F* (task 6) is selected for allocation. Because, first element in the first column of *F* is equal to 2, and it represents task 2. Transition 2 is enabled, but idle time (6) is less than the time of task 2 (10), therefore no allocation is done using first column. In the second column (or second alternative), 1 and 3 are ignored because the tasks having these numbers were allocated to the station and placed into *ASW*. *Step 3.a.i.aa.* and *Step 3.a.i.bb.* prevent reassignment of these tasks. Next value is 6, and task 6 is allocated, because its time (6) is equal to idle time of station 1 (6). *AL* is created and added in *ALS*. New token condition is specified, but next task in *d* (task 8) is not allocated to the station 1, because idle time of the station is equal to 0. Thus, in the beginning of the second iteration, third column of *F* is investigated for initial condition. Although task 8 is selected it isn't allocated according to the token condition. Last column of *F* shows that task 7 is the candidate task to be allocated. But task 7 is not allocated because t_7 is not enabled. Thus only one alternative is created in iteration 2, and task 6 is allocated to station 1. *Sw* is equal to 35 time units, *w* is equal to 1, and new token condition is calculated by adding sixth row of the incidence matrix to token condition in the end of the iteration 1. In the third iteration there isn't any allocation because idle time of station 1 is equal to 0. So *ALS* is empty, *Step 3.b.* is implemented and a new empty station is opened (*w*=2, and S*w*=35). Initial conditions for iteration 4 are *w*=2, *Sw*=35, and the token condition which is equal to in the end of the iteration 3. Task 2, task 4, and task 7 are allocated according to the initial conditions using the first column of *F*. After the allocation of task 7, *Swnow* is equal to 17. Task 9 is selected according to the token condition and *d*, but its time (20) is greater than the current idle time of station 2 (18). Therefore task 9 is not allocated and initial conditions are updated for the second column. Tasks 1, 3, and 6 were allocated in the previous iterations. Therefore following task number, 8, in the list is selected. According to the token condition and current idle time, task 8 is allocated to station 2 (*Swnow*=13). Also the last task, task 10, is appropriate. After the allocation of task 10 to station 2, *Swnow* is equal to 27, and idle time of station 2 is 8. New token condition is calculated for a new allocation. But, since there isn't any value in the second column of *F*, third column is investigated. Tasks 1, 3, and 5 are again ignored, since they were allocated before. Next tasks are candidates for allocation. Since the task numbers and values in the previous column are the same, similar operations are done for task 8 and task 10. Algorithm passes to fourth column because all values in third column were allocated. However any allocation is not done for fourth column, because task 7, is not enabled. All alternatives are listed in *ALS*. Second and third alternatives have the minimum value in the last column of *ALS*. According to the minimum value (8), task 8 and task 10 are allocated to second station. *Sw* is equal to 27, station number *w* is equal to 2, and in the end of the iteration 3, new token condition is calculated by adding eighth and tenth row of incidence matrix to token condition. In the next iteration, only one allocation is generated, because there is no task to be allocated in the second and third columns of *F*, and task 7, which is the third in the fourth column of *F*, is not enabled. Only first value in the first column is available for allocation, task 2, according to the token condition and time constraint. Because time of task 2 and current idle time of station 2 are equal (8).

After the allocation, *Swnow* is equal to 35, and new token is updated using second row of incidence matrix. In the iteration 6, any allocation isn't done because there isn't idle time in station 2 therefore *ALS* is empty. Again new station must be opened to continue the allocation process. Station 3 is opened in iteration 7, and only first column of *F* is used to allocations because task 7, which are the first available task in fourth column, is not allocated according to the token condition and there is no task to allocate in second and third alternative sequences. Task 2, which is the first value in the first column of *F*, was allocated in the previous iterations therefore it is ignored. Second candidate in the list is task 4 according to the token condition and idle time. After the allocation task 4, task 7 is selected and allocated to station 3, because idle time and token condition are available for this allocation process. *Swnow* is equal to 9 time units when the allocating task 4 and task 7 to the station 3. Task 9, which is last value in the list, is allocated to the station 3 because its time (20) is less than the idle time of station 3 (26) and token condition satisfies this allocation. All allocated task numbers are put into *ASW* in the end of the each allocation. After the allocation of task 9, ten tasks are allocated to stations and they are listed in *ASW*. *Step 3* is finished when the number of allocated tasks (*awr*) arrives to number of tasks in the assembly line (*tn*).

Step 4 in the algorithm displays the results like as in previous algorithms. *SON* matrix and line efficiency are given as follows:

w	tn	t	Bz
1	1	10	25
1	3	12	13
1	5	7	6
1	6	6	0
2	8	13	22
2	10	14	8
2	2	8	0
3	4	5	30
3	7	4	26
3	9	20	6

Line efficiency of assembly (LE) = 0.9429

4.7 Conclusion

Proposed five solution algorithms based on Petri nets for SALBP-1 were explained. Complexities of these algorithms were analyzed, and a problem was solved using these algorithms.

In the next chapter, these algorithms will be demonstrated on the benchmarking data, Hoffmann's and Talbot's data sets. Algorithms are coded in MATLAB, the language of technical computing. The results of the solutions, and comparisons with existing algorithms will be given.

CHAPTER FIVE
APPLICATION

5.1 Introduction

In the previous chapter, proposed algorithms based on Petri nets to solve SALBP-1 were explained in detail. In the current chapter, algorithms will be tested on the benchmarking data and compared with other approaches in the literature.

Steps of application procedure will be given in the following section. In the third section, characteristics of data set will be presented. Before the implementation of the algorithms, Petri net models must be created from precedence diagrams of the problems. Characteristics of the Petri net models will be given in section four. In the fifth section, test problems will be solved using proposed algorithms. Performance of the algorithms will be interpreted according to the some classifications such as problem size and order strength. Sixth section will include comparisons between proposed algorithms and existing algorithms and heuristics, and they will be presented into two subsections; in the first subsection, proposed algorithms will be compared with heuristics in QS, and in the second subsection, existing algorithms and heuristics in the literature will be compared the algorithms.

5.2 Application Steps

The proposed five algorithms will be tested on Talbot's and Hoffmann's problem sets (http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/alb/index.htm). When a SALBP-1 is solved by using an algorithm proposed, following steps are put forward.

Step 1. Draw the precedence diagram of tasks.

Step 2. Build the PN model of precedence diagram.

Step 3. Check the Petri net model.

Step 4. Obtain the incidence matrix of PN model.

Step 5. Check the incidence matrix

Step 6. Compute *P*-invariants of the PN model

Step 7. Determine inputs of the problem

Step 8. Solve the problem using the proposed algorithm coded in MATLAB, language of technical computing.

In the first step, precedence diagram of tasks is drawn using problem data. In the next step, PN model of the precedence diagram is built. In this model, tasks are represented by transitions. Therefore, in the Petri net model, number of transitions is equal to the number of tasks in the problem, and number of places is equal to the number of relations in the precedence diagram plus number of tasks not having any successor or predecessor task. Then incidence matrix is obtained from the net. Row numbers in the incidence matrix correspond to the transition numbers, and sum of values in the i_{th} row must be equal to difference between numbers of output places of the i_{th} transition and numbers of input places of the i_{th} transition. Also, column numbers in the incidence matrix correspond to place numbers. If sum of the values in the j_{th} column is -1, it is concluded that the j_{th} place is a source place in the net. Otherwise, if sum of the values in the j_{th} column is 1, the j_{th} place is a sink place. Sum of the values in the other columns must be zero Because of that other places always have only one input and one output transition. Then, *P*-invariants are found. In this study, *P*-invariants were found using DNAnet, a Petri net software tool, up to 70 tasks. For more complicated problems, a MATLAB code (Hanzelek, 2000) was used. Incidence matrix, and time vector, which includes the task times, and cycle time are the inputs of all algorithms. *P*-invariants are also extra inputs for the last algorithm. In the solution step, MATLAB program of the algorithm is run. These programs are given in appendices through 1-5.

5.3 Problem Sets

In this thesis, to test the proposed algorithms, Talbot's and Hoffmann's benchmarking problem sets were used. Talbot's set includes 12 different problems having tasks in varying numbers between 8 and 111, and each problem is combined with several cycle times to build a total of 64 instances. Hoffmann's set includes 5 different problems having tasks in varying numbers between 30 and 111 and totally 50 instances. Thirteen of 50 instances in the Hoffmann's set and 13 of 64 instances in the Talbot's set are the same. Information about these problems is organized in table 5.1. The first and the second column show the author of the problem and the number of tasks, respectively. The third and fourth columns give the minimum and maximum cycle times. The number of different cycle times for each set is given in the fifth column. In this column, T defines a problem in the Talbot's set, H defines a problem in the Hoffmann's set. Sixth and seventh columns contain the minimum and the maximum task times in each problem, respectively. Sum of the task times in each problem is given in the eighth column. Last column shows the order strengths of the problems in percent. Order strength is a measure of the volume of distinct orderings that are permitted by the specified precedence relations (Erel & Sarin, 1998). For example, Tonge's problem includes 70 tasks. Minimum and maximum task times in the problem are 156 and 3510, respectively. Total of task times is 3510. There are 17 instances with different cycle times changing between 160 and 527. Five of these instances have been in Talbot's set, others in Hoffmann's data set.

Table 5.1 Problem Characteristics

Author	# of tasks	Min. cycle time	Max. cycle time	Set	Min. task time	Max. task time	Sum of task times	Order strength (%)
Mertens	7	6	18	T(6)	1	6	29	52.4
Bowman	8	20	20	T(1)	3	17	75	75.0
Jaeschke	9	6	18	T(5)	1	6	37	83.3
Jackson	11	7	21	T(6)	1	7	46	58.2
Mansor	11	48	94	T(3)	2	45	185	60.0
Mitchell	21	14	39	T(6)	1	13	105	71.0
Heskiaof	28	138	342	T(6)	1	108	1024	22.5
Sawyer	30	25	75	H(7),T(7)	1	25	324	44.8
Kilbridge& Wester	45	56	184	T(6),H(6)	3	55	552	44.6
Tonge	70	160	527	T(5),H(12)	1	156	3510	59.4
Arcus1	83	3786	10816	T(7),H(11)	233	3691	75707	59.1
Arcus2	111	5755	17067	T(6),H(14)	10	5689	150399	40.4

5.4 Petri Net Model of the Precedence Diagram

For each problem, building Petri net model of the precedence diagram is the second step. This model will be used in the next steps to define inputs of the solution algorithm. In this study, totally twelve Petri net models were built. Characteristics of these models are given in table 5.2.

Table 5.2 Characteristics of the Petri Net Models

Author	# of tasks	# of transitions	# of places	# of P-invariants
Mertens	7	7	10	3
Bowman	8	8	11	3
Jaeschke	9	9	13	6
Jackson	11	11	15	4
Mansor	11	11	15	4
Mitchell	21	21	31	19
Heskiaof	28	28	42	14
Sawyer	30	30	40	18
Kilbridge&Wester	45	45	70	123
Tonge	70	70	100	853
Arcus1	83	83	115	444
Arcus2	111	111	178	1665

First two columns of table 5.1 and 5.2 are the same. Third and fourth columns display number of transitions, and number of places in the net, respectively. Number of rows and number of columns of the incidence matrix are equal to the number of transitions and number of places, respectively. In other words, a dimension of the incidence matrix is the number of transitions multiplied by the number of places. Also dimensions of time vector and initial token condition vector are related with the information in the third and fourth columns. Number of columns of the time vector is equal to the number of transitions, and number of columns of the initial token condition vector is equal to the number of places. Last column includes number of *P*-invariants of the Petri net model. For example, Petri net model of the Tonge's problem includes 70 transitions and 100 places. Therefore its incidence matrix has 70 rows and 100 columns. Furthermore it has 853 *P*-invariants. Its time vector consists of 70 columns, and its initial token condition vector has 100 columns.

5.5 Results

In this section, table 5.3 and table 5.4 show the solution results of the test problems given in Talbot's and Hoffmann's. Fourth column gives the optimum solutions (number of stations) found by other algorithms previously presented in the literature. Results obtained by the proposed Reachability Analysis Algorithm (RA), Firing Sequences of Transitions Algorithm (FS), Firing Sequences of Transitions-First Search Algorithm (FS-f), Firing Sequences of Transitions-Best Search Algorithm (FS-b), and *P*-invariants Algorithm (P-in) are given in the last five columns.

Table 5.3 The Results for Talbot's Problem Set

Author	Cycle time	# of tasks	Opt.	RA	FS	FS-f	FS-b	P-in
Bowman	20	8	5	5	5	5	5	5
Mansoor	48	11	4	4	5	5	5	4
Mansoor	62	11	3	4	4	4	4	3
Mansoor	94	11	2	2	3	3	3	3
Mertens	6	7	6	6	6	6	6	6
Mertens	7	7	5	5	6	6	6	5
Mertens	8	7	5	5	6	6	6	5
Mertens	10	7	3	4	4	4	4	4
Mertens	15	7	2	2	2	2	2	2
Mertens	18	7	2	2	2	2	2	2
Jaeschke	6	9	8	8	8	8	8	8
Jaeschke	7	9	7	7	7	7	7	7
Jaeschke	8	9	6	6	7	6	6	6
Jaeschke	10	9	4	4	4	4	4	5
Jaeschke	18	9	3	3	3	3	3	3
Jackson	7	11	8	8	9	9	9	8
Jackson	9	11	6	6	6	6	6	6
Jackson	10	11	5	6	6	6	6	6
Jackson	13	11	4	4	5	5	5	4
Jackson	14	11	4	4	4	4	4	4
Jackson	21	11	3	3	3	3	3	3
Mitchell	14	21	8	9	10	10	10	9
Mitchell	15	21	8	8	10	10	10	9
Mitchell	21	21	5	5	6	6	6	6
Mitchell	26	21	5	5	5	5	5	5
Mitchell	35	21	3	3	4	4	4	3
Mitchell	39	21	3	3	3	3	3	3

Table 5.3 (Cont.)

Author	Cycle time	# of tasks	Opt.	RA	FS	FS-f	FS-b	P-in
Heskiaoff	138	28	8	8	9	8	8	8
Heskiaoff	205	28	5	6	6	6	6	6
Heskiaoff	216	28	5	5	6	6	6	5
Heskiaoff	256	28	4	5	5	5	5	5
Heskiaoff	324	28	4	4	4	4	4	4
Heskiaoff	342	28	3	4	4	4	4	3
Sawyer	25	30	14	14	17	15	15	15
Sawyer	27	30	13	13	16	15	14	14
Sawyer	30	30	12	13	13	13	13	12
Sawyer	36	30	10	10	11	11	11	10
Sawyer	41	30	8	9	10	10	10	9
Sawyer	54	30	7	7	7	7	7	7
Sawyer	75	30	5	5	5	5	5	5
Kilbridge	57	45	10	10	12	11	11	10
Kilbridge	79	45	7	8	8	8	8	8
Kilbridge	92	45	6	7	7	7	7	7
Kilbridge	110	45	6	6	6	6	6	6
Kilbridge	138	45	4	5	5	5	5	5
Kilbridge	184	45	3	3	4	4	4	4
Tonge	176	70	21	22	26	24	24	23
Tonge	364	70	10	10	12	11	11	10
Tonge	410	70	9	9	10	10	10	9
Tonge	468	70	8	8	8	8	8	8
Tonge	527	70	7	7	8	8	8	7
Arcus1	5048	83	16	16	17	17	17	16
Arcus1	5853	83	14	14	14	14	14	14
Arcus1	6842	83	12	12	13	13	13	12
Arcus1	7571	83	11	11	11	11	11	11
Arcus1	8412	83	10	10	10	10	10	10
Arcus1	8898	83	9	9	9	9	9	9
Arcus1	10816	83	8	8	8	8	8	8
Arcus2	5755	111	27	27	33	31	31	27
Arcus2	8847	111	18	18	20	19	19	18
Arcus2	10027	111	16	16	18	17	17	16
Arcus2	10743	111	15	15	17	17	17	15
Arcus2	11378	111	14	14	15	15	15	14
Arcus2	17067	111	9	9	10	10	10	9

Table 5.4 The Results for Hoffmann's Problem Set

Author	Cycle time	# of tasks	Opt.	RA	FS	FS-f	FS-b	P-in
Sawyer	27	30	13	13	16	15	14	14
Sawyer	30	30	12	13	13	13	13	12
Sawyer	33	30	11	11	13	13	13	11
Sawyer	36	30	10	10	11	11	11	10
Sawyer	41	30	8	9	10	10	10	9
Sawyer	47	30	7	8	9	9	9	8
Sawyer	54	30	7	7	7	7	7	7
Kilbridge	56	45	10	11	12	11	11	10
Kilbridge	62	45	9	10	11	10	10	10
Kilbridge	69	45	8	9	10	10	10	9
Kilbridge	79	45	7	8	8	8	8	8
Kilbridge	92	45	6	7	7	7	7	7
Kilbridge	111	45	5	6	6	6	6	5
Tonge	160	70	23	23	27	27	27	25
Tonge	168	70	22	23	26	25	25	23
Tonge	176	70	21	22	26	24	24	23
Tonge	185	70	20	21	24	23	23	20
Tonge	195	70	19	20	23	22	22	19
Tonge	207	70	18	18	21	21	21	18
Tonge	220	70	17	17	19	18	18	18
Tonge	234	70	16	16	18	18	18	16
Tonge	251	70	14	15	16	16	16	15
Tonge	270	70	14	14	15	15	15	14
Tonge	293	70	13	13	14	14	14	13
Tonge	320	70	11	12	13	12	12	12
Arcus1	3786	83	21	22	24	23	23	22
Arcus1	3985	83	20	21	22	22	22	21
Arcus1	4206	83	19	20	21	21	21	21
Arcus1	4454	83	18	18	20	20	20	18
Arcus1	4732	83	17	17	19	18	18	17
Arcus1	5048	83	16	16	17	17	17	16
Arcus1	5408	83	15	16	16	15	15	15
Arcus1	5824	83	14	14	14	14	14	14
Arcus1	6309	83	13	13	14	13	13	13
Arcus1	6883	83	12	12	13	13	13	12
Arcus1	7571	83	11	11	11	11	11	11
Arcus2	5785	111	27	27	32	30	30	27
Arcus2	6016	111	26	26	32	30	30	26
Arcus2	6267	111	25	25	31	27	27	26
Arcus2	6540	111	24	25	30	27	27	25
Arcus2	6837	111	23	24	28	26	26	24
Arcus2	7162	111	22	22	26	24	24	23
Arcus2	7520	111	21	22	26	24	24	21

Table 5.4 (Cont.)

Author	Cycle time	# of tasks	Opt.	RA	FS	FS-f	FS-b	P-in
Arcus2	7916	111	20	20	26	24	24	21
Arcus2	8356	111	19	19	24	21	21	20
Arcus2	8847	111	18	18	20	19	19	18
Arcus2	9400	111	17	17	19	18	18	17
Arcus2	10027	111	16	16	18	17	17	16
Arcus2	10743	111	15	15	17	17	17	15
Arcus2	11570	111	13	14	15	15	15	14

All results are summarized in table 5.5 according to the some performance measures. The performance measures examined are number of optimum solutions (# Opt.), number of optimum solutions in % (% Opt.), average deviation (Av. dev.), percentage of average deviation (Av. dev. %), maximum deviation (Max. dev.), and percentage of maximum deviation (Max. dev. %). Number of optimum solutions is number of instances for which an optimal solution is found. Average deviation means average relative deviation from optimality, and maximum deviation means maximum deviation from optimality (Scholl & Klein, 1999). The formulations of these performance measures are shown below where algorithm solution gives the result (number of stations) found by each proposed algorithm for each instance and optimum solution gives the optimum number of stations for each instances:

$$Opt.\% = \frac{\textit{number of optimum solutions for procedure}}{\textit{number of instances}}$$

$$Av.dev. = \frac{\sum(\text{procedure solution - optimum solution})}{\text{number of instances}}$$

$$Av.dev.\% = \frac{\sum 100 \times \left(\frac{(\textit{procedure solution - optimum solution})}{\textit{optimum solution}}\right)}{\textit{number of instances}}$$

$$Max.dev. = \max(\text{procedure solution - optimum solution})$$

$$Max.dev.\% = \max\left[100 \times \left(\frac{(procedure\ solution - optimum\ solution)}{optimum\ solution}\right)\right]$$

In table 5.5, results are classified into three groups such as Talbot's set, Hoffmann's set, and the combination of the two sets. In the end of the comparisons for each problem set, average ranks of the algorithms are given in a row. To compute average ranks, for each performance measure, except % Opt, algorithms are ranked regard to their performances, and a rank of 1 is assigned to the best performer. This is repeated for each performance measure and then to each algorithm the average of the ranks is assigned. Ranking for % Opt. is ignored because % Opt. and # Opt. give the same information. When the ranks of two or more algorithms for the same performance measure are the same, the average of ranks that would have been assigned if the ranks were distinguishable are assigned to each.

Table 5.5 shows that RA is the best performer for all problem sets. *P*-in follows the RA. Firing Sequences of Transitions algorithm and its versions performed worse. FS-f and FS-b almost showed equal performance except that FS-b gives less average deviation than FS-f.

It can be seen that number of optimum solutions in Hoffmann's set is less than those in Talbot's set. In this set, RA is better than P-in for average deviation and maximum deviation, whereas P-in is better than RA for average deviation in % and maximum deviation. Since they have same average ranks they share first rank. FS and its versions are inefficient for the Hoffmann's set. They optimally solved about %40 of the problems in Talbot's set, whereas they solved about % 10 of the problems in Hoffman's set.

Proposed algorithms show similar performances in the Combined set, too. However, the difference between the performances of RA and P-in, and FS and its versions are too much. Although only RA solves 12 instances optimally and only P-in optimally solves 8 instances FS and its versions has no instance which is solved optimally by only FS, only FS-f, or only FS-b.

Table 5.5 The Results for Problem Sets

Data Set	Perf. Meas.	RA	FS	FS-f	FS-b	P-in
Talbot (64 inst.)	*# Opt.*	51	23	25	25	47
	Opt. %	79.688	35.938	39.063	39.0625	73.438
	Av. dev.	0.2031	0.9688	0.7656	0.75	0.2813
	Av. dev. %	4.0476	12.943	11.382	11.2619	5.3825
	Max. dev.	1	6	4	4	2
	Max. dev. %	33.333	50	50	50	50
	Av. rank	**1**	**4.4**	**3.2**	**2.8**	**2**
Hoffmann (50 inst.)	*# Opt.*	27	3	5	5	27
	Opt. %	54	6	10	10	54
	Av. dev.	0.46	2.54	1.76	1.74	0.52
	Av. dev. %	3.963	15.6	11.5	11.3	3.78
	Max. dev.	1	6	4	4	2
	Max. dev. %	20	30	28.6	28.6	16.7
	Av. rank	**1.4**	**5**	**3.4**	**3**	**1.4**
Combined (101 inst.)	*# Opt.*	70	24	28	28	66
	Opt. %	69.30693	23.76238	27.72277	27.72277	65.34653
	Av. dev.	0.3069	1.663366	1.19802	1.188119	0.3762
	Av. dev. %	3.9668	14.31031	11.55427	11.47811	4.6835
	Max. dev.	1	6	4	4	2
	Max. dev. %	33.33333	50	50	50	50
	Av. rank	**1**	**4.4**	**3.2**	**2.8**	**2**

In the literature, size of the problem (i.e. number of tasks), too, is considered in the classifications of the test problems. Therefore, in this study, the problems are classified into three groups according to the their size. Number of tasks in the small sized problems is at most 21; in the medium sized problems it changes between 22 and 45, and in the large sized problems it is more than 45 (Erel *et al.*, 2001). Table 5.6 shows the solution results arranged with respect to the size of the problems, and the performance measures. Number of instances included in small sized group is 21, in medium sized group is 31, and in large sized group is 49.

Table 5.6 shows that all RA outcomes are better than others. As in the former comparisons, performance of P-in is very close to RA. Their performances are better for small and large sized problems. FS again showed worst performance. As the problem size gets larger, FS, Fs-f, and FS-b don't give good results.

Table 5.6 The Results for Problems Given the Size of the Problems

Prob. Size	Perf. Meas.	RA	FS	FS-f	FS-b	P-in
	# Opt.	18	11	12	12	17
	Opt. %	85.71429	52.38095	57.14286	57.14286	80.95238
Small	*Av. dev.*	0.142857	0.47619	0.428571	0.428571	0.190476
(21 inst.)	*Av. dev. %*	4.126984	12.18254	11.3888	11.3888	6.111111
	Max. dev.	1	1	1	1	1
	Max. dev. %	33.33333	50	50	50	50
	Av. rank	**1**	**3.6**	**2.6**	**2.4**	**1.8**
	# Opt.	17	6	7	7	16
	Opt. %	54.83871	19.35484	22.58065	22.58065	51.6129
Medium	*Av. dev.*	0.451613	1.225806	1	0.967742	0.483871
(31 inst.)	*Av. dev. %*	7.597286	17.58925	15.47347	15.22533	7.887668
	Max. dev.	1	3	2	2	1
	Max. dev. %	33.33333	33.33333	33.33333	33.33333	33.33333
	Av. rank	**1**	**4.2**	**3**	**2.6**	**1.6**
	# Opt.	35	7	9	9	33
	Opt. %	71.42857	14.28571	18.36735	18.36735	67.34694
Large	*Av. dev.*	0.28571	2.44898	1.653061	1.653061	0.38776
(49 inst.)	*Av. dev. %*	1.504139	13.14778	9.145664	9.145664	2.04459
	Max. dev.	1	6	4	4	2
	Max. dev. %	9.09091	30	20	20	10.5263
	Av. rank	**1**	**5**	**3**	**3**	**2**

In the literature, order strengths of the problems are considered in the classifications of the test problems. In many research works, usually used order strengths are <0.1, 0.1- 0.5, and > 0.5. In this study, order strengths of test problems are classified as <=50% and >50%. 42 instances of the 101 instances fall into first class, and 59 instances fall into second class. The results for problem sets are given in Table 5.7.

For problems having order strength <=50%, RA algorithm solves optimally 25 of 42 instances. P-in algorithm optimally solves 23 instances. FS, FS-f, and FS-b algorithms find only 4, 5, and 5 optimal solutions, respectively. Ranking does not change according to their average deviations. For av. dev. %, P-in gives better results than the others. But, av. dev. % value of RA is very close to that of P-in. FS and its versions gave the worst results for this performance measure. RA and P-in algorithms have equal performance in maximum deviation. All algorithms gave same max. dev. % (i.e. 33.333).

Table 5.7 The Results for Problems Given the Order Strengths of the Problems

Order Str.	Perf. Meas.	RA	FS	FS-f	FS-b	P-in
	# Opt.	25	4	5	5	23
	Opt. %	59.52381	9.52381	11.90476	11.90476	54.7619
<=50%	*Av. dev.*	0.404762	2.33333	1.52381	1.5	0.452381
(42 ins.)	*Av. dev. %*	5.809157	18.02726	13.56574	13.38259	5.584122
	Max. dev.	1	6	4	4	1
	Max. dev. %	33.33333	33.33333	33.33333	33.333	33.3333
	Av. rank	**1.2**	**4.2**	**3**	**2.6**	**1.4**
	# Opt.	45	20	23	23	43
	Opt. %	76.27119	33.89831	38.98305	38.98305	72.88136
>50%	*Av. dev.*	0.2373	1.186441	0.966102	0.966102	0.32203
(59 ins.)	*Av. dev. %*	2.6553	11.66424	10.12239	10.12239	4.0424
	Max. dev.	1	5	4	4	2
	Max. dev. %	33.33333	50	50	50	50
	Av. rank	**1**	**4.4**	**2.8**	**2.8**	**2**

For problems having order strength >50%, RA is again the best performer for all performance measures, and P-in again follows the RA. When the order strength of the problems increased, number of problems solved optimally increased. FS, FS-f, and FS-b found much more optimum solutions than the first case, but maximum deviation in % values are very high.

In this section, test results were analyzed with respect to problem set, problem size and order strength. RA was the best in all comparisons. All algorithms showed better performance in Talbot's data set for problems with small size, and for problems having order strength >=50%. We can arrange the algorithms from the best one to the worst as follow: RA, P-in, FS-b, FS-f, and FS. All comparisons are summarized in table 5.8, and average ranking of each proposed algorithm is given in parenthesis.

5.6 Comparisons

In the previous section, the algorithms are compared with each other. In this section, these algorithms are compared with existing algorithms from the literature.

Table 5.8 Rankings of Proposed Algorithms in Comparison with Each Other

Classes	Subclasses	1st	2nd	3rd	4th	5th
Data Set	*Talbot's*	RA (1)	P-in (2)	FS-b (2.8)	FS-f (3.2)	FS (4.4)
	Hoffmann's	RA and P-in (1.4)		FS-b (3)	FS-f (3.4)	FS (5)
	Combined	RA (1)	P-in (2)	FS-b (2.8)	FS-f (3.2)	FS (4.4)
Problem Size	*Small*	RA (1)	P-in (1.8)	FS-b (2.4)	FS-f (2.6)	FS (3.6)
	Medium	RA (1)	P-in (1.6)	FS-b (2.6)	FS-f (3)	FS (4.2)
	Large	RA (1)	P-in (2)	FS-b and FS-f (3)		FS (5)
Order Strength	*<=50%*	RA (1.2)	P-in (1.4)	FS-b (2.6)	FS-f (3)	FS (4.2)
	>50%	RA (1)	P-in (2)	FS-b and FS-f (2.8)		FS (4.4)

5.6.1 Comparison with the Algorithms in QS' Results

QS (Quant Systems) is a software package developed by Chang & Sullivan (1991). It includes many subprograms to solve various industrial engineering problems in several areas like linear programming, dynamic programming, queuing theory, projects scheduling, forecasting, scheduling, inventory control, facility layout, and etc. One of the subprograms is Production Line Balancing (PLB). It includes 12 heuristics to solve SALBP-1, up to 1000 tasks. One of them is "Best Bud Search" (BBS) heuristic, an optimization method developed by Nevins in 1972. It does not attempt to minimize the number of stations directly; instead, an upper bound on the number of stations is imposed and the problem is solved for that many stations. If the attempt is successful, the number of stations is decremented by one, and another attempt is made until it is either impossible or computationally impractical to get a smaller number of stations (Erel & Sarin, 1998). Another heuristic is the COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) developed by Arcus in 1966. The main idea in this heuristic is the random generation of feasible sequence and keeping the best sequence. In other ten heuristics in PLB, a sequencing of tasks is created according to the decision rule and first available task in the sequence (primary heuristics) is assigned. If the first task in the list is not available, an available task is selected among the other available tasks according to the decision rule (tie breaker heuristic). To obtain a sequence list, "Fewest Followers" (FF) and

"Most Followers" (MF) use the total number of tasks following each task. In the "Fewest Immediate Followers" (FIF) and "Most Immediate Followers" (MIF), sequencing is obtained using the total number of tasks which must immediately succeed each task. "Longest Process Time" (LPT) and "Shortest Process Time" (SPT) use task times to create a sequence. Although, in FF, FIF, and SPT heuristics, the total number of tasks or task times is arranged in ascending order, they are arranged in descending order in MF, MIF, and LPT heuristics. "Ranked Positional Weight Method" (RPWM) was developed by Helgeson & Birnie in 1961. In this heuristic, task is given a weight value, which is equal to sum of its task time and task times of all succeeding tasks. Tasks are arranged in descending order of weights of tasks. Other three heuristics use the task order number, and sequence lists are obtained according to the task order number. In "First to Become Available" (FBA) heuristic, assignment starts from the first task on the list. If this task is available, it is assigned. Otherwise, second task is investigated for assignment. In "Last to Become Available" (LBA), last task in the list is selected to be assigned. If it is available, it is assigned. If not, searching will continue to backward. If heuristic is "Random" (Rand.) task selection is random If selected task is available, it is allocated to station. Otherwise, another task is selected randomly. Each of FF, MF, FIF, MIF, LPT, SPT, and RPWM heuristics can be selected as primary heuristic and tie breaker heuristic. Thus 6! times combinations can be generated in PLB. In this study, when test problems are being solved using each of these heuristics, that heuristic will be considered as both primary heuristic and tie breaker heuristic. In running of LBA, FBA, and Random heuristics, tie breaker heuristic is not used. Test problems are solved using all heuristics in PLB. Totally 1212 instances are solved using QS. Results of QS are given in appendix 6 and 7. Comparison between QS and proposed algorithms are given in table 5.9.

In Talbot's problem set, COMSOAL solved 57, BBS solved 53, and. RA solved 51 of 64 instances, optimally. P-in algorithm follows them and LPT, and its place with RPWM in ranking is fifth into totally 17 algorithms. FS-f and FS-b gave better results than only SPT. FS and SPT gave same number of optimum solutions. RA has rank 2 for max. dev. %, and rank 3 for av. dev. %. P-in has ranks varying between fifth to eighth. FS and its versions always performed worse than the others. For the

Table 5.9 Comparison with QS According to the Data Sets

		QS												*PROPOSED ALG.*				
Data Set	**Perf. Meas.**	**BBS**	**FF**	**FIF**	**FBA**	**LBA**	**LPT**	**MF**	**MIF**	**Rand.**	**RPWM**	**SPT**	**COM.**	**RA**	**FS**	**FS-f**	**FS-b**	**P-in**
	# Opt.	53	29	29	35	34	50	43	42	32	47	23	57	51	23	25	25	47
	Opt. %	82.8	45.3	45	54.7	53.1	78.1	67.19	66	50	73.438	35.938	89.1	79.688	35.938	39.063	39.063	73.438
Talbot	*Av. dev.*	0.17	0.72	0.6	0.52	0.55	0.22	0.328	0.4	0.547	0.2656	0.9688	0.11	0.2031	0.9688	0.7656	0.75	0.2813
(64 inst.)	*Av .dev. %*	3.15	10.5	9.9	9.3	7.39	4.59	6.017	6.4	8.906	6.0019	12.844	1.32	4.0476	12.943	11.382	11.262	5.3825
	Max. dev.	1	4	3	2	3	1	1	3	2	1	7	1	1	6	4	4	2
	Max.dev. %	33.3	50	50	50	33.3	33.3	33.33	50	33.33	50	50	20	33.333	50	50	50	50
	Av. rank	**1.8**	**11.8**	**10.8**	**8.8**	**8.4**	**3**	**4.8**	**8.4**	**8**	**5**	**14.8**	**1**	**2.4**	**14.6**	**13**	**12.6**	**6.2**
	# Opt.	33	7	7	10	9	27	28	13	7	32	4	28	27	3	5	5	27
	Opt. %	66	14	14	20	18	54	56	26	14	64	8	56	54	6	10	10	54
Hoffmann	*Av. dev.*	0.34	1.74	1.38	1.1	1.3	0.46	0.44	1.02	1.22	0.36	2.48	0.44	0.46	2.54	1.76	1.74	0.52
(50 inst.)	*Av. dev. %*	2.995	10.8	9.1	7.17	8.35	3.99	3.35	6.56	8.0547	3.0181	15.27	3.26	3.963	15.6	11.5	11.3	3.78
	Max. dev.	1	4	4	5	3	1	1	3	3	1	8	1	1	6	4	4	2
	Max.dev. %	16.7	25	25	25	20	20	16.7	16.7	20	16. 7	30.77	16.7	20	30	28.6	28.6	16.7
	Av. rank	**1**	**11.8**	**11.4**	**10.6**	**9.4**	**5.4**	**2.4**	**6.6**	**9.2**	**1.6**	**16.4**	**2.2**	**4.8**	**16.6**	**13.8**	**13.2**	**5**
	# Opt.	76	34	34	40	39	68	64	49	35	70	25	76	70	24	28	28	66
	Opt. %	75.25	33.7	33.66	39.6	38.61	67.33	63.4	48.51	34.65	69.307	24.752	75.248	69.307	23.762	27.723	27.723	65.347
Combined	*Av. dev.*	0.248	1.16	0.95	0.772	0.871	0.327	0.37	0.673	0.851	0.3069	1.6337	0.2475	0.3069	1.6634	1.1980	1.1881	0.3762
(101 inst.)	*Av. dev. %*	3.05	10.8	9.552	8.536	7.825	4.366	4.83	6.503	8.748	4.7909	14.094	1.9730	3.9668	14.310	11.554	11.478	4.6835
	Max. dev.	1	4	4	3	3	1	1	3	3	1	8	1	1	6	4	4	2
	Max.dev. %	33.33	50	50	50	33.33	33.33	33.3	50	33.33	50	50	20	33.333	50	50	50	50
	Av. rank	**1.6**	**11.6**	**11.2**	**8.8**	**8**	**3.4**	**4.6**	**8**	**8.4**	**4.2**	**14.6**	**1**	**2.4**	**15**	**12.8**	**12.4**	**6.6**

Hoffmann's problem set, BBS solved 33, RPWM solved 32 of 50 instances optimally. Number of optimum solutions for MF and COMSOAL are the same. LPT, RA and P-in have rank 5. SPT, FS and its versions showed the worst performance in number of optimum solutions, average deviation and average deviation in percent. For maximum deviation RA, BBS, LPT, MF, RPWM, and COMSOAL produced the best outcomes. For max. dev. %, P-in, BBS, MF, MIF, RPWM, and COMSOAL have rank 1. SPT has the worst outcomes for this performance measure. When the all instances are combined, COMSOAL and BBS have rank 1 for #opt. and max. dev. RA showed its best performance in max. dev. For other performance measures, RA follows COMSOAL and BBS. P-in algorithm is the fifth for av. dev. %, sixth for # opt., seventh for av. dev. and max. dev., and eighth for max. dev. %. FS, FS-f, FS-b, and SPT have the worst outcomes, again.

Also comparison is enlarged using size of problems and order strengths. Results of comparison are given table 5.10 and 5.11. According to the problem size, for small and medium sized problems, COMSOAL was superior to the others for all performance measures. For large sized problems, BBS has the best results. While RA has better performance for large sized problems, P-in was slightly better for medium sized problems. For only small and large sized problems, FS-f and FS-b have better results than SPT and FS. When the order strengths of problems are considered, for the first group, RPWM followed by COMSOAL and BBS was superior to the others. RA is the fourth, and P-in is the seventh with respect to the average. FS-b and FS-f are better than SPT, FS, and FF regard to average ranks. FS is better then only SPT. In the second group of order strength, COMSOAL produced best result for all performance measures. RA is the third, and P-in is the seventh according to the average ranks.

Comparisons have shown that RA has performed well. In many comparisons, it was very close to the best performer. In some comparisons, P-in, too, has outcomes as good as the best algorithm has. FS, FS-f, and FS-b have the worst performance in many comparisons. Comparisons are summarized in table 5.12, and average ranks of the algorithms are given in parenthesis.

Table 5.10 Comparison with QS According to the Problem Size

		QS												*PROPOSED ALG.*				
Prob. Size	**Perf. Meas.**	**BBS**	**FF**	**FIF**	**FBA**	**LBA**	**LPT**	**MF**	**MIF**	**Rand.**	**RPWM**	**SPT**	**COM.**	**RA**	**FS**	**FS-f**	**FS-b**	**P-in**
Small (21 inst.)	*# Opt.*	19	15	15	15	18	18	15	19	13	13	11	21	18	11	12	12	17
	Opt. %	90.48	71.4	71.43	71.43	85.71	85.71	71.4	90.48	61.9	61.905	52.381	100	85.714	52.381	57.143	57.143	80.952
	Av. dev.	0.095	0.29	0.286	0.286	0.143	0.143	0.29	0.095	0.381	0.381	0.4762	0	0.1429	0.4762	0.4286	0.4286	0.1905
	Av. dev. %	2.54	8.65	8.651	10.08	3.492	4.127	6.71	3.571	9.001	10.198	12.183	0	4.1260	12.183	11.554	11.478	6.1111
	Max. dev.	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
	Max.dev%.	33.33	50	50	50	33.33	33.33	33.3	50	33.33	50	50	0	33.333	50	50	50	50
	Av. rank	**2**	**7**	**7**	**7.6**	**3**	**3.4**	**5.6**	**3.6**	**7.8**	**9.4**	**11.6**	**1**	**3.4**	**11.6**	**10.6**	**10.4**	**6.2**
Medium (31 inst.)	*# Opt.*	18	6	7	11	9	16	16	11	7	19	6	23	17	6	7	7	16
	Opt. %	58.06	19.4	22.58	35.48	29.03	51.61	51.6	35.48	22.58	61.29	19.355	74.194	54.839	19.355	22.581	22.581	51.613
	Av. dev.	0.419	0.97	0.903	0.71	0.742	0.484	0.48	0.677	0.806	0.3871	1.1936	0.2581	0.4516	1.2258	1	0.9677	0.4839
	Av. dev. %	6.334	15.3	14.41	12.11	12.25	8.592	8.47	11.07	13.4	6.7049	17.287	3.6726	7.5973	17.590	15.474	15.225	7.8877
	Max. dev.	1	3	2	2	3	1	1	2	3	1	4	1	1	3	2	2	1
	Max.dev. %	25	40	33.33	33.33	33.33	33.33	33.3	33.33	33.33	33.333	33.333	20	33.333	33.333	33.333	33.333	33.333
	Av. rank	**2.2**	**14.6**	**9.2**	**7.4**	**9.2**	**4.4**	**4**	**7**	**9.8**	**2.2**	**13.4**	**1**	**3.2**	**13**	**10.4**	**9.6**	**4**
Large (49 inst.)	*# Opt.*	39	13	12	14	12	34	33	19	15	38	8	32	35	7	9	9	33
	Opt. %	79.59	26.5	24.49	28.57	24.49	69.39	67.3	38.78	30.61	77.551	16.327	65.306	71.429	14.286	18.367	18.367	67.347
	Av. dev.	0.204	1.65	1.265	1.02	1.265	0.306	0.33	0.918	1.082	0.2245	2.4082	0.3469	0.2857	2.4490	1.6531	1.6531	0.3878
	Av. dev. %	1.192	8.79	6.866	5.616	6.884	1.794	1.73	4.869	5.697	1.2626	12.893	1.7432	1.5041	13.148	9.1457	9.1457	2.0446
	Max. dev.	1	4	1	3	3	1	1	3	3	1	8	1	1	6	4	4	2
	Max.dev. %	9.091	19	17.39	14.29	15.79	9.091	9.091	16.67	14.29	9.091	30.769	9.091	9.091	30	20	20	10.526
	Av. rank	**1**	**12.6**	**9.4**	**9**	**10.6**	**3.2**	**3.2**	**8.8**	**9.2**	**1.6**	**16.4**	**4**	**2.2**	**16.6**	**13.8**	**13.8**	**6.8**

Table 5.11 Comparison with QS According to the Order Strength

		QS												*PROPOSED ALG.*				
Order Str.	**Perf. Meas.**	**BBS**	**FF**	**FIF**	**FBA**	**LBA**	**LPT**	**MF**	**MIF**	**Rand.**	**RPWM**	**SPT**	**COM.**	**RA**	**FS**	**FS-f**	**FS-b**	**P-in**
	# Opt.	28	4	5	10	9	24	25	12	6	29	4	25	25	4	5	5	23
	Opt. %	66.67	9.52	11.9	23.81	21.43	57.14	59.5	28.57	14.29	69.048	9.5238	59.524	59.524	9.5238	11.905	11.905	54.762
<=%50	*Av. dev.*	0.333	1.52	1.357	1.024	1.071	0.429	0.43	0.857	1.095	0.3095	2.2143	0.4048	0.4048	2.3333	1.5238	1.5	0.4524
(42 ins.)	*Av. dev. %*	4.466	13.6	12.51	9.743	10.2	6.047	5.53	8.734	10.73	4.049	17.284	3.5028	5.8092	18.027	13.566	13.383	5.5841
	Max. dev.	1	4	4	3	3	1	1	2	3	1	8	1	1	6	4	4	1
	Max.dev. %	25	40	33.33	33.33	33.33	33.33	33.3	33.33	33.33	33.333	33.333	20	33.333	33.333	33.333	33.333	33.333
	Av. rank	**2**	**14.6**	**10.2**	**7.8**	**8.4**	**4.4**	**3.4**	**7**	**9**	**1.6**	**13.4**	**1.8**	**3.2**	**13.6**	**11.2**	**10.6**	**4.6**
	# Opt.	48	30	29	30	30	44	51	37	29	41	21	51	45	20	23	23	43
	Opt. %	81.36	81.4	81.36	81.36	81.36	81.36	81.4	81.36	81.36	81.356	81.356	81.356	76.271	33.898	38.983	38.983	72.881
>%50	*Av. dev.*	0.186	0.9	0.661	0.593	0.729	0.254	0.32	0.542	0.678	0.3051	1.2203	0.1356	0.2373	1.1864	0.9661	0.9661	0.3220
(59 ins.)	*Av. dev. %*	2.043	8.75	7.45	7.677	6.135	3.169	4.34	4.915	7.338	5.3191	11.824	0.8839	2.6553	11.664	10.122	10.122	4.0424
	Max. dev.	1	4	3	2	3	1	1	3	3	1	5	1	1	5	4	4	2
	Max.dev%.	33.33	50	50	50	33.33	33.33	33.3	50	33.33	50	50	12.5	33.333	50	50	50	50
	Av. rank	**2**	**11.2**	**10**	**9**	**8.2**	**3.2**	**3.2**	**8**	**8.8**	**5.8**	**15**	**1**	**2.6**	**14.6**	**12.6**	**12.6**	**6.6**

Table 5.12 Rankings in comparison with QS

Classes	Subclasses	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	17th
Data Set	*Talbot's*	COMSOAL (1)	BBS (1.8)	**RA (2.4)**	LPT (3)	MF (4.8)	RPWM (5)	**P-in (6.2)**	Random (8)	LBA and MIF (8.4)		FBA (8.8)	FIF (10.8)	FF (11.8)	**FS-b (12.6)**	**FS-f (13)**	**FS (14.6)**	SPT (14.8)
	Hoffmann's	BBS (1)	RPWM (1.6)	COMSOAL (2.2)	MF (2.4)	**RA (4.8)**	**P-in and LPT (5)**		MIF (6.6)	Random (9.2)	LBA (9.4)	FBA (10.6)	FIF (11.4)	FF (11.8)	**FS-b (13.2)**	**FS-f (13.8)**	SPT (16.4)	**FS (16.6**
	Combined	COMSOAL (1)	BBS (1.6)	**RA (2.4)**	LPT (3.4)	RPWM (4.2)	MF (4.6)	**P-in (6.6)**	LBA and MIF (8)		Random (8.4)	FBA (8.8)	FIF (11.2)	FF (11.6)	**FS-b (12.4)**	**FS-f (12.8)**	SPT (14.6)	**FS (15)**
Problem Size	*Small*	COMSOAL (1)	BBS (2)	LBA (3)	**RA and LPT (3.4)**		MIF (3.6)	MF (5.6)	**P-in (6.2)**	FF and FIF (7)		FBA (7.6)	Random (7.8)	RPWM (9.4)	**FS-b (10.4)**	**FS-f (10.6)**	SPT and **FS (11.6)**	
	Medium	COMSOAL (1)	BBS and RPWM (2.2)		**RA (3.2)**	**P-in** and MF **(4)**		LPT (4.4)	MIF (7)	FBA (7.4)	FIF and LBA (9.2)		**FS-b (9.6)**	Random (9.8)	**FS-f (10.4)**	**FS (13)**	SPT (13.4)	FF (14.6)
	Large	BBS (1)	RPWM (1.6)	**RA (2.2)**	MF and LPT (3.2)		COMSOAL (4)	**P-in (6.8)**	MIF (8.8)	FBA (9)	Random (9.2)	FIF (9.4)	LBA (10.6)	FF (12.6)	**FS-f and FS-b (13.8)**		SPT (16.4)	**FS (16.6)**
Order Strength	*<=50%*	RPWM (1.6)	COMSOAL (1.8)	BBS (2)	**RA (3.2)**	MF (3.4)	LPT (4.4)	**P-in (4.6)**	MIF (7)	FBA (7.8)	LBA (8.4)	Random (9)	FIF (10.2)	**FS-b (10.6)**	**FS-f (11.2)**	SPT (13.4)	**FS (13.6)**	FF (14.6)
	>50%	COMSOAL (1)	BBS (2)	**RA (2.6)**	LPT and MF (3.2)		RPWM (5.8)	**P-in (6.6)**	MIF (8)	LBA (8.2)	Random (8.8)	FBA (9)	FIF (10)	FF (11.2)	**FS-f and FS-b (12.6)**		**FS (14.6)**	SPT (15)

5.6.2 Comparison with Other Methods

In this section, proposed algorithms are compared with existing heuristics, branch and bound algorithms, tabu search metaheuristics, and genetic algorithms. This section is classified into four subsections according to the solution approaches in these studies.

5.6.2.1 Comparison with LBHA-1 Heuristic

In the literature, many heuristics exist to solve SALBP-1. Some of them were given in section 5.6.1. LBHA-1 was developed by Baybars (1986b). It includes five phases. In the first phase, the size of the problem is reduced by eliminating certain tasks. Phase 2 identifies mutually exclusive assignment decisions. In the third phase, the problem is decomposed into smaller problems. In phase 4, any collections of tasks likely to be assigned to the same station are identified and, finally, in phase 5 a solution is sought using various heuristics for the network/problem. Baybars solved 52 instances which were obtained from Mertens', Bowman's, Jaeschke's, Jackson's, Mitchell's, Heskiaoff's, Sawyer's, Kilbridge & Wester's, and Arcus' problems. LBHA-1 provides a feasible solution when the cycle time is less than maximum task time in the problem by opening the parallel stations. Three instances were solved for this condition by LBHA-1. Therefore, since the proposed algorithms consider the serial assembly lines, outcomes of three instances are ignored in the comparison. Solution results, obtained by LBHA-1 and the developed algorithms, of other 49 instances are given in appendix 8. Optimum solutions for six instances are different in study of Baybars (1986b) and in study of Scholl & Klein (1997). Also optimum solutions are shown in appendix 8 according to the Baybars and Scholl & Klein. Comparison with LBHA-1 is given in table 5.13. In this comparison, Scholl's optimum solutions are used as optimum solution.

In this comparison LBHA-1 has the best outcomes for # Opt., % Opt., Av. dev., and Av. dev. %. It optimally solved 40 instances, RA and P-in optimally solved 38 and 35 instances, respectively. LBHA-1 individually found optimum solutions for

Table 5.13 Comparison with LBHA-1

Perf. Meas.	LBHA-1	RA	FS	FS-f	FS-b	P-in
# Opt.	40	38	22	23	23	35
Opt. %	81.633	77.5510	44.898	46.939	46.939	71.43
Av. dev.	0.2	0.22	0.76	0.64	0.6	0.286
Av. dev. %	3.848	4.434	10.964	9.838	9.523	5.492
Max. dev.	2	1	3	3	3	1
Max. dev. %	66.667	33.333	42.857	42.857	42.857	33.333
Av. rank	**2.2**	**1.6**	**5**	**4.2**	**3.8**	**2.4**

five instances, whereas at least one of the proposed algorithms individually found optimum solutions for four instances. Outcomes of LBHA-1 and RA for average deviation and average deviation in percent are very close to each other. RA is the best performer for maximum deviation. Proposed algorithms also found better solutions than LBHA-1 for Max. dev %. FS, FS-f, and FS-b found worst solutions with respect to all performance measures except Max. dev %. They are better than LBHA-1 for only % Max. dev.

Comparison results are also classified by problem size and order strength, and given in table 5.14 and 5.15. LBHA-1 found better solutions for small sized problems. It was succeeded by RA, FS-f, FS-b, FS and P-in according to the average ranks. RA was the best for medium sized problems. For large sized problems, LBHA-1, RA, and P-in have equal performance. When the classification by order strength is considered, RA is the best algorithm for first order strength group. RA and P-in are better than LBHA-1 with respect to average ranks. In the second group of order strength, LBHA-1 has better performance than proposed algorithms. RA and P-in follow it in average ranking.

In the light of observations above, it can be concluded that LBHA-1 is the best for small sized problems and second order strength group. RA algorithm is the best for medium sized problems and first order strength group. LBHA-1, RA and P-in have equal performance for large sized problems. P-in is better than LBHA-1 for medium sized problems and first order strength group. FS-b has same rank with LBHA-1 for medium sized problems, and FS-f and FS-b are better than P-in for small sized problems. Comparisons are summarized in table 5.16.

Table 5.14 Comparison with LBHA-1 According to the Problem Size

Problem Size	Perf. Meas.	LBHA-1	RA	FS	FS-f	FS-b	P-in
Small (16 inst.)	*# Opt.*	16	13	11	12	12	12
	Opt. %	100	81.25	68.75	75	75	75
	Av. dev.	0	0.188	0.3125	0.25	0.25	0.25
	Av. dev. %	0	4.896	6.8304	5.7887	5.7887	6.4583
	Max. dev.	0	1	1	1	1	1
	Max. dev. %	0	33.33	33.333	33.333	33.333	33.333
	Av. rank	**1**	**1.6**	**4**	**2.2**	**2.2**	**2.6**
Medium (26 inst.)	*# Opt.*	17	18	6	6	6	16
	Opt. %	65.385	69.23	23.077	23.077	23.077	61.538
	Av. dev.	0.3846	0.308	1.1923	1	0.9231	0.3846
	Av. dev. %	7.3992	5.514	16.321	14.795	14.203	6.3764
	Max. dev.	2	1	3	2	2	1
	Max. dev. %	66.667	33.33	33.333	33.333	33.333	33.333
	Av. rank	**3.2**	**1**	**4.6**	**3.6**	**3.2**	**1.8**
Large (7 inst.)	*# Opt.*	7	7	5	5	5	7
	Opt. %	100	100	71.429	71.429	71.429	100
	Av. dev.	0	0	0.2857	0.2857	0.2857	0
	Av. dev. %	0	0	2.0833	2.0833	2.0833	0
	Max. dev.	0	0	1	1	1	0
	Max. dev. %	0	0	8.3333	8.3333	8.3333	0
	Av. rank	**1**	**1**	**4**	**4**	**4**	**1**

Table 5.15 Comparison with LBHA-1 According to the Order Strength

Order Strength	Perf. Meas.	LBHA-1	RA	FS	FS-f	FS-b	P-in
<=50% (21 ins.)	*# Opt.*	13	13	4	4	4	12
	Opt. %	61.905	61.905	19	19	19.05	57.143
	Av. dev.	0.429	0.381	1.33	1.1	1	0.429
	Av. dev. %	7.576	6.828	17	15.1	14.41	7.101
	Max. dev.	2	1	3	2	2	1
	Max. dev. %	66.667	33.333	33.333	33.333	33.333	33.333
	Av. rank	**3**	**1**	**4.6**	**3.6**	**3.2**	**1.8**
>50% (28 ins.)	*# Opt.*	27	25	18	19	19	23
	Opt. %	96.429	89.289	64.3	67.9	67.86	82.143
	Av. dev.	0.036	0.107	0.36	0.32	0.321	0.1791
	Av. dev. %	1.191	2.7989	6.8	6.21	6.21	4.286
	Max. dev.	1	1	1	1	1	1
	Max. dev. %	33.333	33.333	33.333	33.333	33.333	33.333
	Av. rank	**1**	**1.6**	**4**	**2.8**	**2.8**	**2.2**

Table 5.16 Rankings in Comparison with LBHA-1

Classes	Subclasses	1st	2nd	3rd	4th	5th	6th
Data Set		**RA (1.6)**	LBHA-1 (2.2)	**P-in (2.4)**	**FS-b (3.8)**	**FS-f (4.2)**	**FS (5)**
Problem Size	*Small*	LBHA-1 (1)	**RA (1.6)**	**FS-f and FS-b (2.2)**		**P-in (2.6)**	**FS (4)**
	Medium	**RA (1)**	**P-in (1.8)**	**FS-b and** LBHA-1 **(3.2)**		**FS-f (3.6)**	**FS (4.6)**
	Large	**RA, P-in, and LBHA-1 (1)**			**FS, FS-f, and FS-b (4)**		
Order Strength	*<=50%*	**RA (1)**	**P-in (1.8)**	LBHA-1 (3)	**FS-b (3.2)**	**FS-f (3.6)**	**FS (4.6)**
	>50%	LBHA-1 (1)	**RA (1.6)**	**P-in (2.2)**	**FS-f and FS-b (2.8)**		**FS (4)**

5.6.2.2 Comparison with Algorithms based on Branch and Bound Procedures

Branch and bound algorithms are widely used for solving combinatorial problems. As its name implies, the approach consists of two fundamental procedures. Branching is the process of partitioning a large problem into two or more subproblems, and bounding is the process of calculating a lower bound on the optimal solution of given subproblem. (Baker, 1974, p.55). Many algorithms based on the branch and bound procedures have been developed to solve SALBP-1. FABLE, OptPack, EUREKA, SALOME, and AGSA are well-known effective algorithms.

FABLE (Fast Algorithm for Balancing Lines Effectively) was developed by Johnson (1988). It is a single-solution method and performs forward planning. Tasks are relabeled, and the nodes of branch and bound tree relate to tasks. An alternative is created and subsequently used. This search is called laser search. FABLE does not use any additional heuristic in branching process. Its lower bounds are LB1, bin packing and one-machine bounds for bounding process. Maximum Load Rule (Jackson-1 Dominance), the Labeling Dominance from Schrage & Baker in 1978, Jackson-2 Dominance, the First Station Dominance, and the Task Duration Incrementing Rule are used for logical tests. (Scholl & Klein, 1999) (Sprecher, 1999).

OptPack was developed by Nourie & Venta (1991). Its characteristics for branching process are same with FABLE. In additional it uses packing and Hoffmann heuristic in branching stage. Only LB1 is used for bounding process, and logical tests are done using Maximum Load Rule and Tree Dominance Rule. (Scholl & Klein, 1999).

EUREKA, which is a single solution method, was developed by Hoffmann (1992). It is first method uses the forward and backward planning. The method is station-oriented although first two methods are task-oriented. In branching stage, lower bound method and Hoffmann heuristic are employed. Also original task numbers are used, and task numbers are not relabeled. In bounding stage LB1 is used. EUREKA does not use a logical test. (Scholl & Klein, 1999), (Sprecher, 1999).

Scholl & Klein (1997) developed SALOME, which is a multiple-solution method. The method employs bidirectional search. Tasks are relabeled and both directions are simultaneously searched. Local lower bound method, which was developed by them, is used to decide the planning direction for branching process. LB1, bin packing, one- and parallel-machine bounds are used for bounding process. Its logical tests are Maximum Load Rule, Simple Permutation rule, Jackson Dominance Rule, Task Time Incrementing Rule, and Dynamic Prefixing. (Scholl & Klein, 1999) (Sprecher, 1999).

AGSA (Adapted General Sequencing Algorithm) was developed by Sprecher (1999). Method employs forward and backward planning. Tasks are relabeled and an algorithm, which was developed by Sprecher and by adapting resource-constrained project scheduling problem (RCPSP) to SALBP-1. Extended Jackson Rule, Maximum Load Rule, Set-Based Dominance and different solution characteristics are used for branching process and logical test. AGSA uses more than one lower bound by applying Extended Task Duration Incrementing Rule.

Table 5.17 shows outcomes of the algorithms with respect to the performance measures. Average deviation and maximum deviation measures could not be used in this comparison because results of the studies with existing branch and bound algorithms do not include any information related with these performance measures or solution for each instance.

Table 5.17 Comparison with Branch and Bound Algorithms

Data Set	Perf. Meas.	FAB.	Opt.	EUR.	SAL.	AGSA	RA	FS	FS-f	FS-b	P-in
Talbot* *(64 inst.)	*# Opt.*	64	64	64	64	64	51	23	25	25	47
	Opt. %	100	100	100	100	100	79.688	35.938	39.063	39.063	73.438
	Av. dev. %	0	0	0	0	0	4.0476	12.943	11.382	11.262	5.3825
	Max.dev. %	0	0	0	0	0	33.333	50	50	50	50
	Av. rank	**1**	**1**	**1**	**1**	**1**	**6**	**8.667**	**8**	**7.667**	**7**
Hoffmann* *(50 inst.)	*# Opt.*	48	48	47	48	48	28	3	5	5	27
	Opt. %	96	96	94	96	96	56	6	10	10	54
	Av. dev. %	0.25	0.25	0.34	0.25	0.25	3.9	15.6	11.5	11.3	3.78
	Max.dev. %	7.69	7.69	12.00	7.69	7.69	20	30	28.6	28.6	16.7
	Av. rank	**1**	**1**	**5**	**1**	**1**	**6. 667**	**9. 667**	**8.333**	**8**	**6.333**

All of the algorithms based on the branch and bound procedures gave better results than proposed algorithms with respect to all performance measures used. They solved all instances in Talbot's set optimally. As a result, our proposed algorithms produced worse results than the algorithms based on the branch and bound procedures. Rankings of this comparison are given in table 5.18.

Table 5.18 Rankings in Comparison with Branch and Bound Algorithms

Classes	Subclasses	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Data Sets	*Talbot's*	**FABLE, OptPack, EUREKA, SALOME, and AGSA (1)**					**RA (6)**	**P-in (7)**	**FS-b (7.667)**	**FS-f (8)**	**FS (8.667)**
	Hoffmann's	**FABLE, OptPack, SALOME, and AGSA (1)**				**EUREKA (5)**	**P-in (6.333)**	**RA (6.667)**	**FS-b (8)**	**FS-f (8.333)**	**FS (9.667)**

5.6.2.3 Comparison with Algorithms based on Tabu Search Metaheuristics

Tabu search, which was introduced by Glover (1990), is a metaheuristic strategy for solving combinatorial optimization problems. The underlying idea is to forbid some search directions at a present iteration in order to avoid cycling, but to be able to escape from a local optimal point (Chiang, 1998).Many problems, for example employee scheduling, space planning, probabilistic logic problems, job shop scheduling, machine scheduling, traveling salesman problems, and etc., are successfully solved using tabu search (Glover, 1990). Chiang (1998) developed algorithms based on tabu search to solve SALBP-1. They are Best Improvement with Task Aggregation (BI with TA), Best Improvement without Task Aggregation (BI without TA), First Improvement with Task Aggregation (FI with TA), Best Improvement with Task Aggregation (FI without TA). He tested his algorithms on Mertens',Bowman's, Jaeschcke's, Jackson's, Dar-El's, Mithcell's, Heskiaof's, Sawyer's, Kilbridge & Wester's, Tonge', and Arcus' problems (totally 61 instances). All solutions of Changs and ours are given in appendix 9. except solutions of Dar-El's problem, because of that we don't have any information about this problem. A comparison is made between Chang's results and our results in table 5.19 for performance measures.

Table 5.19 Comparison Chang's Algorithms

Perf. Meas.	**BI with TA**	**BI without TA**	**FI with TA**	**FI without TA**	**RA**	**FS**	**FS-f**	**FS-b**	**P-in**
# Opt.	47	58	47	58	49	23	25	25	45
Opt. %	77.049	95.082	77.049	95.082	80.328	37.705	40.984	40.984	73.771
Av. dev.	0.262	0.049	0.262	0.049	0.197	0.967	0.754	0.738	0.279
Av. dev. %	2.804	0.358	2.716	0.463	3.700	11.803	10.166	10.040	4.828
Max. dev.	2	1	2	1	1	6	4	4	2
Max. dev. %	25	10	25	11.111	33.333	33.333	33.333	33.333	33.333
Av. rank	**3.8**	**1**	**3.6**	**1.4**	**3.4**	**8.2**	**7**	**6.6**	**5.4**

BI without TA, and FI without TA solved 58 instances, and RA solved 49 instances, optimally. BI with TA, and FI with TA achieved 47 optimum solutions. P-in follows them with 45 optimum solutions. FS, FS-f, and FS-b have the worst results for # Opt. and for average deviation. Chang's four algorithms have better performance than proposed algorithms for Av.dev. % and Max. dev. %. According to the maximum deviation, RA has equal performance with BI without TA, and FI without TA, and P-in has same performance with BI with TA, and FI with TA. BI without TA is the best performer with respect to average ranks, and FI without TA follows it. RA is worse than BI without TA, and FI without TA but is better than the rest.

Results of comparisons with respect to problem size and order strength are given in table 5.20 and 5.21. In problems with small and medium sized, Chang's algorithms are better than proposed algorithms. However, for large sized problems, RA is the best Although FS, FS-f, and FS-b have outcomes close to those of BI with TA, and FI with TA, they succeed them in average ranking. In the first group of order strength Chang's four algorithms are better than proposed algorithms. In the second group, RA has rank 3, that is, it succeeds BI without TA, and FI without TA. FS, FS-f, and FS-b were placed in the end of average ranking.

Table 5.20 Comparison Chang's Algorithms According to the Problem Size

Problem Size	Perf. Meas.	BI with TA	BI without TA	FI with TA	FI without TA	RA	FS	FS-f	FS-b	P-in
Small (18 inst.)	*# Opt.*	17	18	17	18	16	11	12	12	15
	Opt. %	94.444	100	94.444	100	88.889	61.111	66.667	66.667	83.333
	Av. dev.	0.0556	0	0.0556	0	0.1111	0.3889	0.3333	0.3333	0.1667
	Av. dev. %	0.9259	0	0.9259	0	2.9629	8.1944	7.2685	7.2685	4.3519
	Max. dev.	1	0	1	0	1	1	1	1	1
	Max. dev. %	16.667	0	16.667	0	33.333	33.333	33.333	33.333	33.333
	Av. rank	**3**	**1**	**3**	**1**	**4.6**	**7**	**5.8**	**5.8**	**5.2**
Medium (25 inst.)	*# Opt.*	21	25	21	25	16	6	7	7	13
	Opt. %	84	100	84	100	64	24	28	28	52
	Av. dev.	0.16	0	0.16	0	0.36	1.08	0.88	0.84	0.48
	Av. dev. %	2.7381	0	2.8	0	6.7048	16.451	14.673	14.365	8.2648
	Max. dev.	1	0	1	0	1	3	2	2	1
	Max. dev. %	25	0	25	0	33.333	33.333	33.333	33.333	33.333
	Av. rank	**3**	**1**	**3.2**	**1**	**4.6**	**8.2**	**7**	**6.6**	**5.2**
Large (18 inst.)	*# Opt.*	9	15	9	15	17	6	6	6	17
	Opt. %	50	83.333	50	83.333	94.444	33.333	33.333	33.333	94.444
	Av. dev.	0.6111	0.1667	0.6111	0.1667	0.0556	1.3889	1	1	0.1111
	Av. dev. %	4.7737	1.21142	4.3879	1.56967	0.2646	8.9561	6.8041	6.8041	0.5291
	Max. dev.	2	1	2	1	1	6	4	4	2
	Max. dev. %	12.5	10	12.5	11.1111	4.7619	23.81	14.815	14.815	9.5238
	Av. rank	**5**	**2.6**	**4.8**	**3**	**1**	**8.6**	**7**	**7**	**2.2**

Table 5.21 Comparison Chang's Algorithms According to the Order Strength

Order Strength	Perf. Meas.	BI with TA	BI without TA	FI with TA	FI without TA	RA	FS	FS-f	FS-b	P-in
<=50% (25 ins.)	*# Opt.*	19	24	19	23	17	4	5	5	16
	Opt. %	76	96	76	92	68	16	20	20	64
	Av. dev.	0.24	0.04	0.24	0.08	0.32	1.4	1.04	1	0.36
	Av. dev. %	3.3307	0.22222 2	3.115	0.73016	6.2048	15.415	12.867 5	12.559 8	6.4648
	Max. dev.	1	1	1	1	1	6	4	4	1
	Max. dev.%.	25	5.55556	25	11.1111	33.333	33.333	33.333	33.333	33.333
	Av. rank	2.8	1	2.6	1.8	4.2	8.2	7	6.6	4.8
>50% (36 ins.)	*# Opt.*	28	34	28	35	32	19	20	20	29
	Opt. %	77.778	94.444	77.778	97.2222	88.889	52.778	55.556	55.556	80.556
	Av. dev.	0.2778	0.05556	0.278	0.0278	0.1111	0.6667	0.5556	0.5556	0.2222
	Av. dev. %	2.4383	0.45139	2.4383	0.27778	1.961	9.2951	8.2898	8.2898	3.6905
	Max. dev.	2	1	2	1	1	5	3	3	2
	Max. dev.%	16.667	10	16.667	10	33.333	33.333	33.333	33.333	33.333
	Av. rank	4.2	1.6	4.2	1	3	8.2	6.6	6.6	4.6

Comparisons are summarized in table 5.22.

Table 5.22 Rankings in Comparison with Chang's Algorithms

Classes	Subclasses	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Data Set		BI without TA (1)	FI without TA (1.4)	**RA (3.4)**	FI with TA (3.6)	BI with TA (3.8)	**P-in (5.4)**	**FS-b (6.6)**	**FS-f (7)**	**FS (8.2)**
Problem Size	*Small*	BI without TA and FI without TA (1)		BI with TA and FI with TA (3)		**RA (4.6)**	**P-in (5.2)**	**FS-f and FS-b (5.8)**		**FS (7)**
	Medium	BI without TA and FI without TA (1)		BI with TA (3)	FI with TA (3.2)	**RA (4.6)**	**P-in (5.2)**	**FS-b (6.6)**	**FS-f (7)**	**FS (8.2)**
	Large	**RA (1)**	**P-in (2.2)**	BI without TA (2.6)	FI without TA (3)	FI with TA (4.8)	BI with TA (5)	**FS-f and FS-b (7)**		**FS (8.6)**
Order Strength	*<=50%*	BI without TA (1)	FI without TA (1.8)	FI with TA (2.6)	BI with TA (2.8)	**RA (4.2)**	**P-in (4.8)**	**FS-b (6.6)**	**FS-f (7)**	**FS (8.2)**
	>50%	FI without TA (1)	BI without TA (1.6)	**RA (3)**	BI with TA and FI with TA (4.2)		**P-in (4.6)**	**FS-f and FS-b (6.6)**		**FS (8.2)**

5.6.2.4 Comparison with Genetic Algorithms

Genetic Algorithm (GA) is a search tool for global optimization in complex search space. It starts with an initial population of individuals (also called chromosomes or strings) representing different possible solutions to a problem. The population is maintained by the iterations of the algorithm, called generations. At each generation, the fitness of each individual is evaluated, and stochastically selected for the next generation based upon its fitness. New individuals, called offsprings, are produced by two genetic operators, crossover and mutation. The offsprings are supposed to inherit the good attributes from their parents, so that the average quality of solutions becomes better than in the previous population. This evolutionary process is repeated until some stopping criteria are met. (Kim *et al.*, 1996)

Kim *et al.* (1996) proposed a genetic approach to solve SALBP-1, and used Kilbridge & Wester's 45-task problem having cycle times varying between 57 and 79 (K57CT and K79CT in table 5.23), and Tonge's 70-task problem having cycle times varying between 176 and 320 (T176CT and T320CT in table 5.23). They compared the outcomes with those of four heuristics; minimum upper bound (MIN-UB), maximum task time (MAX-DUR), maximum task time divided by task upper bound (MAX-DUR/UB), and largest set rule (LSR). These test problems are also solved in this study by using the proposed algorithms. Table 5.23 shows the solution results of these algorithms.

The GA found the minimum number of stations for all test problems. Proposed algorithms except FS found same solution with the GA for Tonge's problem with cycle time 320. For Kilbridge's test problem with cycle time 57, RA and P-in gave same results with the GA. GA is better than the proposed algorithms for other test problems. In the majority of test problems, the proposed algorithms showed better performance than the existing heuristics considered here.

Table 5.23 Comparison with Some Heuristics and Kim's Genetic Algorithm

	Heuristics				Kim *et al.* GA		Proposed Algorithms				
Problem	**MAX-DUR**	**MIN-UB**	**MAX-DUR/UB**	**LSR**	**Mean**	**Best**	**RA**	**FS**	**FS-f**	**FS-b**	**P-in**
T176CT	26	25	26	23	21.1	21	22	26	24	24	23
T320CT	13	13	13	12	12	12	12	13	12	12	12
K57CT	11	11	11	11	10	10	10	12	11	11	10
K79CT	8	8	9	8	7.5	7	8	8	8	8	8

A GA was proposed by Sabuncuoglu *et al.* (2000), tested on Tonge's 70 tasks problem, and its outcomes were compared with those of Moodie & Young's heuristic, Tonge's heuristics; MIF, RC, and BPC, Nevins' heuristic, and Baybars' heuristic; LBHA-1. We ignore five cycle times used by Sabuncuoglu *et al.* for parallel assembly lines, since the proposed algorithms consider serial assembly lines. Table 5.24 shows the outcomes of the algorithms used in comparison.

Table 5.24 Comparison with Some Heuristics and Sabuncuoglu's Genetic Algorithm

Cycle time	**Opt. Sol.**	**Moodie & Young**	**Tonge MIF**	**Tonge RC**	**Tonge BPC**	**Nevins**	**Baybars**	**Their GA**	**RA**	**FS**	**FS-f**	**FS-b**	**P-in**
170	22	24	24	24	23	23	23	23	23	26	25	25	23
173	22	24	24	24	23	22	23	23	23	26	25	25	23
176	22	22	24	23	22	22	23	22	22	26	24	24	23
179	21	22	23	23	21	21	22	22	21	26	23	23	21
182	21	22	23	22	21	21	22	22	21	25	23	23	21
346	11	11	11	12	11	11	11	11	11	12	12	12	11
349	11	11	11	11	11	11	11	11	11	12	12	12	11
364	11*	11	11	11	11	11	11	11	10	12	11	11	10

First of all, optimum solution for cycle time equal to 364 is 10 stations, not 11 (11 stations according to the Baybars, 10 stations according to the Scholl). In five instances, RA showed equal performance with GA proposed by Sabuncuoglu *et al.* However, RA is better in the rest of instances. P-in has same outcomes with GA for four instances. But, P-in performed better in three instances. In most instances GA is better than FS, FS-f and FS-b.

RA and P-in found better solutions than the heuristics examined here. Outcomes of RA are not worse than those of other heuristics except one instance. RA and P-in are better than all heuristics for last instance. For only two instances other heuristics have better performance than P-in.

5.7 Conclusion

In this chapter, proposed algorithms were tested on the problems given in Talbot's and Hoffman's data sets. Performances of the proposed algorithms were compared with each other, and with the other algorithms. As a result, RA is the best of proposed algorithms, and P-in algorithm follows RA when the proposed algorithms were compared with each other. According to the comparison between the proposed algorithms and heuristics in QS, RA was better than the most of the heuristics in QS, P-in produced better results than half of the heuristics in QS. Branch and bound procedures were better than the proposed algorithms. When compared with the tabu search metaheuristics, sometimes RA and P-in were performed better. RA and P-in gave better solutions than genetic algorithms in many problems. As a result, especially RA and P-in algorithms performed better than many heuristics. Their performances were good as well as tabu search metaheuristics and genetic algorithms, but they need improvements across to branch and bound procedures

CHAPTER SIX
CONCLUSION AND FUTURE RESEARCH

Simple assembly line balancing problem-type 1 (SALBP-1) involves minimizing the number of stations on an assembly line for a given cycle time. As they are known as NP-hard problems, approximate or heuristic approaches are extremely required for solving these problems.

In this thesis, five heuristic algorithms based on Petri nets (PNs) were developed to solve SALBP-1. As mathematical and graphical tools, PNs have been widely used to design and analyze the discrete event systems. A great deal of research attention on assembly systems has been focused on solving the problems such as modeling, controlling, planning, task sequencing, and scheduling. To the best of our knowledge, this is the first study that uses PNs for solving SALBP-1.

In the thesis, first, a review of the recent research works that applied to SALBP-1, and a review of PNs applications on manufacturing problems and classification frameworks were provided. Then, five algorithms based on Petri nets to solve SALBP-1 were developed. PNs enabled the model building from the precedence diagram of the assembly line considered, and determination of the available tasks by token movement. Complexity analyses of these algorithms were presented, and implementations of the algorithms were explained with the use of an example. All iterations of solutions were given in details.

First algorithm developed uses reachability analysis of PNs. Current token condition in the net specifies firable transitions (i.e. available tasks). The task giving minimum station idle time is selected among the candidates, and assigned to current station. New token condition is found and new candidates are determined considering the new token condition. The second algorithm assigns tasks due to firing sequences of transitions. It doesn't use any decision rule. When the time of a task in the sequence is greater than the idle time of the current station, this task is automatically assigned to the new station. The third, too, uses firing sequences of transitions. It assigns tasks to stations like in the second algorithm until it finds a task, in the firing sequence, having time greater than the idle time of current station. At this point, following tasks are sequentially investigated to find a task whose time is less than or equal to the idle time of the current station. First task satisfying the time constraint is assigned to the current station. If there isn't any task satisfying this condition, then new station is opened automatically. First task in the sequence is assigned to the new station. Assignment procedure of the fourth algorithm is like those of second and third algorithms. When the time of first task in the firing sequence is greater than the idle time of the current station, this algorithm searches appropriate task as the second algorithm. It finds the task giving minimum idle time among candidates and assigns it to the current station although the third algorithm assigns first available task among candidates. When the candidates don't satisfy the time constraint, new station is opened and first task in the sequence is assigned to the new station.. Last algorithm finds alternative routes using *P*-invariants of the Petri net model of the precedence diagram, and temporarily assigns available tasks from the routes. The tasks giving minimum idle time among the temporarily assigned tasks are determined and assigned to station, and new token condition and idle time of station are found after this assignment.

Later, these algorithms were coded in MATLAB, the language of technical computing, and tested on Talbot's and Hoffman's benchmarking problem sets. Solution results were given according to the data sets, problem size, and order strengths. Furthermore, the algorithms were compared with each other considering some performance measures such as number of optimum solutions, percentage of optimum solutions, average deviation, percentage of average deviation, maximum deviation, and percentage of maximum deviation.

Finally, proposed algorithms were also compared with i) heuristics used by QS software package, ii) a heuristic, LBHA-1, iii) FABLE, OptPack, EUREKA, SALOME, and AGSA (branch and bound algorithms based heuristics), iv) four algorithms based on tabu search metaheuristics, v) two heuristics based on genetic algorithms. Same performance measures and problem classification, given above, were used. RA and *P*-in showed equal performance in the solution of problems in Hoffmann's set. Owing to comparison results, proposed algorithms could be arranged according to their performances, from the best one to the worst as follows: Reachability analysis (RA), *P*-invariant (*P*-in), Firing sequence of transitions-best search (FS-b), Firing sequence of transitions-first search (FS-f), and Firing sequence of transitions (FS). According to the comparison between the proposed algorithms and heuristics in QS, RA was better than the most of the heuristics in QS, P-in produced better results than half of the heuristics in QS. Branch and bound procedures were better than the proposed algorithms. When compared with the tabu search metaheuristics, sometimes RA and P-in were performed better. RA and P-in gave better solutions than genetic algorithms in many problems. As a result, especially RA and P-in algorithms performed better than many heuristics. The computational results obtained by RA and *P*-in are very encouraging. They have competitive power across the heuristics and genetic algorithms. However, algorithms based on branch and bound procedures produced better results than the proposed algorithms.

In light of the above observations, there is a need for further research to improve FS, FS-f, and FS-b so that the resulting procedures can effectively search for optimal solution as well as RA and *P*-in procedures, and also to increase quality of the solutions found by the RA and *P*-in compared with the algorithms based on branch and bound procedures. Capabilities of the proposed algorithms can offer benefits to solve other types of assembly line balancing problem in the future.

REFERENCES

Adamou, M., Zerhoni, S. N., & Bourjault, A. (1998). Hierarchical modeling and control of flexible assembly systems using object-oriented Petri nets. International Journal of Computer Integrated Manufacturing. Vol. 11, No. 1, pp. 18-33.

Ajenblit, D. A. & Wainwrigth, R. L. (1998). Applying genetic algorithms to the U-shaped assembly line balancing problem. Proceedings of the IEEE International Conference on Evolutionary Computation, ICEC'98, May 4-9 1998, Anchorage, AK, USA, pp. 96-101

Alpan, G. & Jafari, M. A. (1997). Dynamic analysis of timed Petri nets: a case of two processes and a shared resource. IEEE Transactions on Robotics and Automation. Vol.13, No.3, pp. 338-346.

Askin, R. G. & Standridge, C.R. (1993). Modeling and analysis of manufacturing systems. John Wiley & Sons Inc.

Barkoui, K., Abdallah, B., & Itmi, M. (1994). Performance evaluation of tool management in FMS by Petri net simulation. Proceedings of the European Simulation Symposium, Istanbul, Turkey, pp.75-79.

Bautista, J.,Suarez, R., Mateo, M., & Companys, R. (2000). Local search heuristics for assembly lien balancing problem with incompatibilities between tasks. Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, CA. April 2000. pp. 2404-2409.

Baybars, I (1986a). A survey of exact algorithms for the simple assembly line balancing problem. Management Science. Vol. 32, No. 8, pp. 909-932.

Baybars, I. (1986b). An efficient heuristic method for the simple assembly line balancing problem. International Journal of Production Research. Vol. 24, No. 1, pp. 149-166.

Cao, T. & Sanderson, A.C. (1994). Task decomposition and analysis of robotic assembly task plans using Petri nets. IEEE Transactions on Industrial Electronics, Vol. 41, No.6, pp. 620-629.

Cecil, J. A., Srihari, K., & Emerson, C. R. (1992). A review of Petri net applications in manufacturing. The International Journal of Advanced Manufacturing Technology. No.7, pp. 168-177.

Chiang, W. C. (1998). The application of a tabu search metaheuristic to the assembly line balancing problem. Annals of Operations Research. No. 77, pp. 209-227.

Chu, F. & Xie, X.L. (1997). Deadlock analysis of Petri nets using siphons and mathematical programming. IEEE Transactions on Robotics and Automation. Vol.13, No.6, pp. 793-804.

Corominas, J.P.A. (1999). Modelling and solving the SALB-E problem. Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning. Porto, Portugal, July. pp 356-360.

D'Souza, K. A. & Khator, S. K. (1997). System reconfiguration to avoid deadlocks in automated manufacturing systems. Computers Industrial Engineering, Vol. 32, No.2. pp 455-465.

Desrochers, A. A. & Al-Jaar, R. Y.(1995): Applications of Petri Nets in Manufacturing Systems, Modeling, Control, and Performance Analysis. IEEE Press, New York.

DiCesare, F., Harhalakis, G., Proth, J.M., Silva, M., & Vernadat, F.B. (1993). Practice of Petri Nets in Manufacturing. Chapman&Hall

Erel, E. & Gokcen, H. (1999). Shortest-route formulation of mixed-model assembly line balancing problem. European Journal of Operational Research. No.116, pp.194-204.

Erel, E. & Sarin, S. C. (1998). A survey of the assembly line balancing procedures. Production Planning & Control. Vol. 9, No. 5., pp. 414-434.

Erel, E., Sabuncuoglu, I., & Aksu, B. A. (2001). Balancing U-type assembly systems using simulated annealing. International Journal of Production Research. Vol. 39, No. 13, pp. 3003-3015.

Ezpeleta, J., Colom, J.M., & Martinez, J. (1995). A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Transactions on Robotics and Automation. Vol.11, No.2, pp. 173-184.

Ezpeleta, J. & Colom, M. (1997). Automatic synthesis of colored Petri nets for the control of FMS. IEEE Transactions on Robotics and Automation. Vol.13, No.3, pp. 327-337.

Fokkert, J. I. Z. & Kok, T. G. (1997). The mixed and multi model line balancing problem: a comparison. European Journal of Operational Research. Vol. 100, pp. 399-412.

Frey, G. (2000). Assembly line sequencing based on Petri net t-invariants. Control Engineering Practice. Vol. 8, pp. 63-69.

Glover, F. (1990). Tabu search: A tutorial. INTERFACES. Vol. 20, No. 4, pp. 74-94.

Gokcen, H. & Erel, E. (1997). A goal programming approach to mixed-model assembly line balancing problem. International Journal of Production Economics. Vol. 48, pp. 177-185.

Groover, M. P. (1992). Automation, production systems, and computer integrated manufacturing. Prentice Hall of India Private Limited, New Delphi.

Hanzalek, Z. Matlab Based Petri Net Analysis. In: Matlab 2000 - Sbornik prispevku 8. rocniku konference. Prague : Institute of Chemical Technology, 2000, p. 78-83

Hoffmann, T. R. (1992). EUREKA: A hybrid system for assembly line balancing. Management Science. Vol. 38, pp. 39-47.

Jeng, M. D. (1997). Petri nets for modeling automated manufacturing systems with error recovery. IEEE Transaction on Robotics and Automation. Vol. 13, No. 5, pp. 752-760.

Jeng, M. D., Lin, C. S., & Huang Y. S. (1999). Petri net dynamics-based scheduling of flexible manufacturing systems with assembly. Journal of Intelligent Manufacturing. Vol. 10, pp. 541-555.

Johnson, R. V. (1988). Optimally balancing large assembly lines with FABLE. Management Science. Vol. 34, No. 2, pp. 240-253.

Kim.Y. K., Kim, Y.J., & Kim, Y. (1996). Genetic algorithms for assembly line balancing with various objectives. Computers and Industrial Engineering. Vol. 30, No. 3, pp. 397-409.

Kim, Y.J., Kim, Y.K., & Cho, Y. (1998). A heuristic-based genetic algorithm for workload smoothing in assembly lines. Computers and Operations Research. Vol. 25, No. 2, pp. 99-111.

Kim, Y. K., Kim, J. Y., & Kim, Y. (2000a). A Coevolutionary Algorithm for Balancing and Sequencing in Mixed Model Assembly Lines. Applied Intelligence. Vol. 13, pp. 247-258.

Kim, Y. K., Kim, S. J., & Kim, J. Y. (2000b). Balancing and sequencing mixed-model U-lines with a co-evolutionary algorithm. Production & Planning Control. Vol. 11, No. 8, pp. 754-764.

Kiritsis, D. & Porchet, M. (1996). A generic Petri net model for dynamic process planning and sequence optimization. Advances in Engineering Software. Vol.25, pp.61-71.

Klein, R. & Scholl, A. (1996). Maximizing the production rate in simple assembly line balancing- a branch and bound procedure. European Journal of Operational Research. Vol. 91, pp. 367-385.

Lee, D. Y. & DiCesare, F. (1994a). Scheduling flexible manufacturing systems using Petri nets and heuristic search. IEEE Transactions on Robotics and Automation. Vol. 10, No. 2, pp.123-132.

Lee, D.Y., & DiCesare, F. (1994b): Integrated scheduling of flexible manufacturing systems employing automated guided vehicles. IEEE Transactions on Industrial Electronics, Vol. 41, No.6, pp. 602-609.

Lee K. H. & Jung, M. Y. (1994). Petri net application in flexible process planning. Computer and Industrial Engineering. Vol.27, Nos 1-4, pp. 505-508.

McCarragher B. J. (1994). Petri net modeling for robotic assembly and trajectory planning. IEEE Transactions on Industrial Electronics. Vol.41, No.6, pp.631-640.

Merengo, C., Nava, F., Pozzetti, A. (1999). Balancing and sequencing manual mixed-model assembly lines. International Journal of Production Research. Vol. 37, No. 12, pp.2835-2860.

Mo, J.P.T. & Tang, B.C.K. (1998). Petri net modeling and design of task oriented messaging system for robot control. Computers and Industrial Engineering. Vol.34, No. 4, pp.729-742.

Moore, K. E. & Gupta, S. M. (1996). Petri net models of flexible and automated manufacturing systems: a survey. International Journal of Production Research, Vol.34, No.11, pp.3001-3035.

Moore, K. E. & Gupta, S. M. (1998). Stochastic Coloured Petri net (SCPN) models of traditional and flexible kanban systems. International Journal of Production Research, Vol.37, No.9, pp.2135-2158.

Moore, K. E., Gungor, A., & Gupta, M. S. (1998). A Petri net approach to disassembly process planning. Computers and Industrial Engineering. Vol. 35, Nos1-2, pp. 165-168.

Moore, K. E., Güngör, A., & Gupta, S. M. (2001). Petri net approach to disassembly process planning for products with complex AND/OR precedence relationships. European Journal of Operational Research. Vol. 135, pp. 428-449.

Mullen, P. R. & Fraizer, G. V. A. (1997). A heuristic for solving mixed-model line balancing problems with stochastic task durations and parallel stations. International Journal of Production Economics. Vol. 51, pp. 177-190.

Murata, T. (1989): "Petri Nets: Properties, Analysis and Applications". Proceedings of IEEE, Vol. 77, No. 4, pp.541-580.

Nance, D. W. & Naps, T. L. (1992). Introduction to computer science: programming, problem solving, and data structures. 2nd edition. West Publishing Company

Nourie, F. J. & Venta, E. R. (1991). Finding the optimal line balances with OptPack. Operations Research Letters. Vol.10, pp. 165-171.

Palekar, U. S. (1998). Lecture notes. Facilities Planning & Design. Mechanical & Industrial Engineering. Univesity of Illinois at Urbana-Campaign. http://www.cen.uiuc.edu/courses/ie261/notes/alb/alb-An.html

Peterson, J.L. (1981). Petri Net theory and the modeling of systems. Prentice-Hall, Englewood Cliffs, New Jersey.

Proth, J. M. & Xie, X. (1996). Petri nets , A tool for design and management of manufacturing systems. John Wiley&Sons, Chichester.

Proth, J.M., Sauer, N., & Xie, X. (1997). Optimization of the number of transportation devices in a flexible manufacturing system using event graphs. IEEE Transactions on Industrial Electronics. Vol.44, No.3, pp. 298-306.

Ramaswany, S. & Valavanis, K.P. (1994). Modeling, analysis and simulation of failures in a material handling system with extended Petri nets. IEEE Transactions on Systems, Man, and Cybernetics. Vol.24,No. 9, pp. 1358-1373.

Ramaswamy, S., Valavanis, K.P., & Barber, S. (1997). Petri net extensions for the development of MIMO net models of automated manufacturing systems. Journal of Manufacturing Systems. Vol.16, No.3, pp. 175-191.

Reisig, W. (1992). A Primer in Petri Net Design. Springer-Verlag, Berlin.

Rubinovitz, J. & Levitin, G. (1995). Genetic algorithm for assembly line balancing. International Journal of Production Economics. Vol. 41, pp. 343-354.

Sabuncuoglu, I., Erel, E., & Tanyer, M. (2000). Assembly line balancing using genetic algorithms. Journal of Intelligent Manufacturing. No.11, pp. 295-310.

Saltzman, M. J. & Baybars, I. (1987). A two-process implicit enumeration algorithm for the simple assembly line balancing problem. European Journal of Operational Research. Vol. 32, pp. 118-129.

Scholl, A. & Voss, S. (1996). Simple assembly line balancing-heuristic approaches. Journal of Heuristics. No. 2, pp. 217-244.

Scholl, A. & Klein, R. (1997). SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. INFORMS Journal on Computing. Vol. 9, No. 4, pp. 319-334.

Scholl, A. & Klein, R. (1999a). Balancing assembly lines effectively - A computational comparison. European Journal of Operational Research. No.114, pp.50-58

Scholl, A. & Klein, R. (1999b). ULINO: Optimally balancing U-shaped JIT assembly lines. International Journal of Production Research. Vol. 37, No. 4, pp. 721-736.

Seifodini, H. & Zhang, J. (1996). Application of simulation and Petri net modeling in manufacturing control systems. International Journal of Production Research. Vol. 34, no. 1, pp. 191-207.

Shiizuka, H., & Suzuki, M. (1994). Modeling of AGV networks in flexible manufacturing systems. Computers and Industrial Engineering, Vol. 27, Nos 1-4, pp. 81-86.

Song, J. S. & Lee, T.E. (1998). Petri net modeling and scheduling for cyclic job shops with blocking. Computer and Industrial Engineering. Vol.34, No.2, pp. 281-295.

Sprecher, A. (1999). A competitive branch-and-bound algorithm for the simple assembly line balancing problem. International Journal of Production Research. Vol. 37, No. 8, pp. 1787-1816.

Sun, T.H., Cheng C.W., & Fu, L.C. (1994). A Petri net based approach to modeling and scheduling for an FMS and a case study. IEEE Transactions on Industrial Electronics. Vol. 41, No.6, pp. 593-600.

Talbot, F .B., Patterson, J .H., & Geh, W. V. (1986). A comparative evaluation of heuristic line balancing techniques. Management Science. Vol. 32, No. 4, pp. 430-454.

Talbot, F. B. & Patterson, J. H. (1984). An integer programming algorithm with network cuts for solving the assembly line balancing problem. Management Science. Vol. 30, No. 1, pp. 85-99.

Tang, Y., Zhou, M. C., & Caudill, R. J. (2001). An integrated approach to disassembly planning and demanufacturing operation. IEEE Transactions on Robotics and Automation. Vol. 17, No. 6, pp. 773-783.

Teng, S. H. & Zhang, J. (1993). A Petri net-based decomposition approach in modeling of manufacturing systems. International Journal of Production Research. Vol. 31,No. 6, pp. 1423-1439.

Thomas, J.P., Nissanke, N., & Baker, K.D. (1996). A hierarchical Petri net framework for the representation and analysis of assembly. IEEE Transactions on Robotics and Automation. Vol.12, No.2, pp. 268-279.

Tiwari, M.K., Hazarika, B., Vidyarthi, N.K., Jaggi, P., & Mukhopadhyay, S.K. (1997). A heuristic solution approach to the machine loading problem of an FMS and its Petri net model. International Journal of Production Research. Vol.35, No.8, pp.2269-2284.

Tiwari, M. K., Chandrasekaran M., & Mohanty R. P. (2001). Use of timed Petri net and activity cycle diagram methodologies for modeling tandem AGVs in FMSs and their performance evaluation. International Journal of Computer Integrated Manufacturing, Vol.14, No.4, pp.399-408.

Uğurdağ, H. F., Rachamadugu, R., & Papachristou, C.A. (1997). Designing paced assembly lines with fixed number of stations. European Journal of Operational Research. No. 102, pp. 488-501.

Vairaktarakis, G. L. & Szmerekovsky, J. (2001). Level workforce schedules for 2-stage transfer lines. Technical Reports. Department of Operations, Weatherhead School of Management, Case Western Reserve University

Venkatesh, K. & Ilyas, M. (1993). Modeling, controlling, and simulation of local area networks for flexible manufacturing systems using Petri nets. Proceedings of the 15th Annual Conference on Computers and Industrial Engineering. Computers and Industrial Engineering Vol.25, Nos1-4, pp. 155-158.

Venkatesh, K., Zhou, M. C., & Caudill, R. J. (1994). Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system. IEEE Transactions on Industrial Electronics. Vol. 41, No.6, pp.611-619.

Wang, J. (1998). Timed Petri nets theory and application. Kluver Academic Publishers, Boston.

Wang, L.C. & Wu, S.Y. (1998). Modeling with colored timed object-oriented Petri nets for automated manufacturing systems. Computers and Industrial Engineering. Vol. 34, No.2, pp. 463-480.

Wu, N. (1999). Necessary and sufficient conditions for deadlock-free operation in flexible manufacturing systems using a colored Petri net. IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews. Vol.29, No.2, pp. 192-204.

Yan, H., Wang, N., Cui, X., & Zhang, J. (1997). Modeling, scheduling, and control of flexible manufacturing systems by extended high-level evaluation Petri nets. IIE Transactions, Vol.29, pp. 147-158.

Yasuda, G. & Tachibana, K. (1994). A parallel distributed control architecture for multiple robot systems using a network of microcomputers. Computers and Industrial Engineering. Vol. 27, Nos1-4, pp. 63-66.

Yee, S. T. & Ventura, J. A. (1999). A Petri net model to determine optimal assembly sequences with assembly operation constraints. Journal of Manufacturing Systems. Vol. 18, No. 3, pp. 203-213.

Yeung, W. H. R. & Moore, P. R. (2000). Genetic algorithms and flexible process planning in the design of fault tolerant cell control for flexible assembly systems. International Journal of Computer Integrated Manufacturing. Vol. 13, No. 2, pp. 157-168.

Zha, X. F., Lim, S. Y. E., & Fok, S. C. (1998). Integrated knowledge-based Petri net intelligent flexible assembly planning. Journal of Intelligent Manufacturing. Vol.9, pp. 235-250.

Zha, X. F. (2000). An object-oriented knowledge based Petri net approach to intelligent integration of design and assembly planning. Artificial Intelligence in Engineering. Vol. 14, pp. 83-112.

Zha, X. F. & Lim, S. Y. E. (2000). Assembly/disassembly task planning and simulation using expert Petri nets. International Journal of Production Research. Vol. 38, No. 15, pp. 3639-3676.

Zha, X. F., Du, H., & Lim, Y. E. (2001). Knowledge intensive Petri net framework for concurrent intelligent design of automatic assembly systems. Robotics and Computer Integrated Manufacturing. Vol. 17, pp. 379-398.

Zhou, M.C. & DiCesare, F. (1993). Petri net synthesis for discrete event control of manufacturing systems. Kluver Academic Publishers, Boston.

Zhou, M. C., McDermott, K., & Patel, P. A. (1993). Petri net synthesis and analysis of a flexible manufacturing system cell. IEEE Transactions on Systems, Man, and Cybernetics. Vol.23, No.2, pp. 523-531.

Zhou, M. C., & Venkatesh, K. (1999). Modeling, simulation, and control of flexible manufacturing systems; A Petri net approach. World Scientific Publishing Co. Pte. Ltd., Singapore.

Zimmermann, A. & Hommel, G. (1999). Modeling and evaluation of manufacturing systems using dedicated Petri nets. The International Journal of Advanced Manufacturing Technology. No. 15, pp. 132-138.

Zuberek, W.M. & Kubiak, W. (1994a). Modeling simple schedules of manufacturing cells using timed Petri nets. Proceedings of the IEEE International Workshop on Intelligent Systems and Innovative Computations, The 6-th Bellman Continuum, Tokyo, Japan, August 1-3, pp.38-47.

Zuberek, W.M. & Kubiak, W. (1994b). Throughput analysis of manufacturing cells using timed Petri nets. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, San Antonio, Texas, October 2-5, pp.1328-1333.

Zuberek, W.M. (1995). Schedules of flexible manufacturing cells and their timed colored Petri net models. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada, October 22-25, pp. 2142-2147.

Zuberek, W.M. (1996). Composite schedules of manufacturing cells and their timed Petri net models. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Beijing, China, October 14-17, pp.2990-2995.

Zuberek, W. M. (1998). Hierarchical derivation of schedules for manufacturing systems. 9-th Symp. On Information Control in Manufacturing; Nancy-Metz, France, June 24-26, pp 423-428.

Zuberek, W.M. & Kubiak, W. (1999). Timed Petri nets in modeling and analysis of simple schedules for manufacturing cells. Computers and Mathematics with Applications. Vol. 37, pp.191-206.

Zurawski, R., & Zhou, M. (1994). Petri nets and industrial applications: A Tutorial. IEEE Transactions on Industrial Electronics, Vol. 41, No. 6, pp. 567-581

Zussman, E. & Zhou, M. C. (1999). A methodology for modeling and adaptive planning of disassembly processes. IEEE Transaction on Robotics and Automation. Vol, 15, No. 1, pp.190-194.

DNAnet, A Petri Net Concurrent Systems Modelling Tool, (C)1995 Data Network Architectures Laboratory, University of Cape Town

MATLAB, The Language of Technical Computing Version 6.1.0.450 Release 12.1, May 18, 2001, Copyright 1984-2001 The MathWork, Inc.

QS (Quant System) Version 2.1. Yih-Long Chang and Robert S. Sullivan. Copyright Prentice-Hall, Inc., 1990, 1991

APPENDICES

Appendix 1. MATLAB Code for Reachability Analysis Procedure

```
clear
disp('This program solves SALBP-1using RA')
% A incidence matrix
A=input('input the incidence matrix in the brackets')
[tn,pn]=size(A)
% M initial token conditions
M=input('input initial token conditions in vector format in the brackets')
% t vector which includes times of jobs
t=input('input times of jobs in vector format in the brackets')
t=t'
Sumt=sum(t)
% C cycle time
C=input('input cycle time')
w=1; Sw=0; Bz=C; minBz=C; SumOt=0; MT=[]; MY=[]; MN=[]; Ot=0; Oz=0; O=[];
SON=[]; a=0
while Sumt>SumOt
 for j=1:tn
   for i=1:pn
     if A(j,i)==-1
       y=M(1,i)
       MT=[MT;y]
     end
   end
   if all(MT==1)
     g=j
     B=A(g,:)
     MY=[MY;B]
     Q=[g t(g)]
     O=[O;Q]
   end
   MT=[]
 end
 [vr,vc]=size(O)
 for v=1:vr
  if O(v,2)==C-Sw & a~=1
    minBz=0;MN=MY(v,:);Oz=O(v,1);Ot=O(v,2);SN=[w Oz Ot minBz];a=1
  end
  if O(v,2)<C-Sw & a~=1
```

```
    Bz=C-Sw-O(v,2)
     if Bz<minBz & Bz>0
       minBz=Bz;MN=MY(v,:);Oz=O(v,1);Ot=O(v,2);SN=[w Oz Ot minBz]
     end
   end
  end
  a=0
  SON=[SON;SN]
  tim=SON(:,3)
  SumOt=sum(tim)
  Sw=C-minBz
  if Oz==[]
   w=w+1;Bz=C;minBz=C;Sw=0; MN=zeros(1,pn)
  end
  M=M+MN
  MN=[]
  MY=[]
  O=[]
  Oz=[]
  Ot=[]
  B=[]
  SN=[]
end
disp('   St.  Job  Time  Idle')
disp('   ---  ---  ----  ----')
disp(SON)
pause
LE=Sumt/(w*C)
disp('Line efficency of assembly is(LE)');disp(LE)
```

Appendix 2. MATLAB Code for Firing Sequencing of Transitions Procedure

```
clear
disp('This program solves SALBP-1using FS')
% A incidence matrix
A=input('input the incidence matrix in the brackets')
[tn,pn]=size(A)
% M initial token conditions
M=input('input initial token conditions in vector format in the brackets')
% t vector which includes times of jobs
t=input('input times of jobs in vector format in the brackets')
t=t'
Sumt=sum(t)
% C cycle time
C=input('input cycle time')
w=1;Bz=C;time=0;mint=sum(t)+1;SumOt=0;MT=[];MN=[];Oz=0;O=[];SEQ=[];
NSEQ=[];SN=[];SON=[];MB=M
sr=0;GT=zeros(1,1);yg=0
while tn>sr
for j=1:tn
    for i=1:pn
      if A(j,i)==-1
        y=M(1,i)
        MT=[MT;y]
      end
    end
    if all(MT==1)
      g=j
      [s1,s2]=size(GT)
      for s=1:s1
        if g==GT(s)
          yg=1
        end
      end
      if yg~=1
        yt(g)=time+t(g)
        Q=[g yt(g)]
        O=[O;Q]
      end
    end
    MT=[];yg=0
 end
 [vr,vc]=size(O)
 for v=1:vr
   if O(v,2)<mint
     mint=O(v,2);Oz=O(v,1);r=v
   end
 end
 time=mint;MN=A(Oz,:);M=M+MN;mint=Sumt
```

```
  SEQ=[SEQ;Oz];
  [sr,sc]=size(SEQ)
  UO=O(1:r-1,:);AO=O(r+1:vr,:);O=[UO;AO]
  if O==[]
    GT=zeros(1,1)
  else
    GT=O(:,1)
  end
end
NSEQ=SEQ;M=MB;MT=[]
while Sumt>SumOt
  if Bz==0 | Bz<t(NSEQ(1,:))
    Bz=C;w=w+1;SN=[]
  end
  if Bz>=t(NSEQ(1,:))
    Bz=Bz-t(NSEQ(1,:));SumOt=SumOt+t(NSEQ(1,:))
    [nsr,nsq]=size(NSEQ);M=M+A(NSEQ(1),:)
    SN=[w NSEQ(1,:) t(NSEQ(1,:)) Bz];NSEQ=NSEQ(2:nsr,:)
  end
SON=[SON;SN]
end
disp('   St.  Job  Time  Idle')
disp('   ---  ---  ----  ----')
disp(SON)
pause
LE=Sumt/(w*C)
disp('Line efficency of assembly is(LE)');disp(LE)
```

Appendix 3. MATLAB Code for Firing Sequencing of Transitions-First Search Procedure

```
clear
disp('This program solves SALBP-1using FS-f')
% A incidence matrix
A=input('input the incidence matrix in the brackets')
[tn,pn]=size(A)
% M initial token conditions
M=input('input initial token conditions in vector format in the brackets')
% t vector which includes times of jobs
t=input('input times of jobs in vector format in the brackets')
t=t'
Sumt=sum(t)
% C cycle time
C=input('input cycle time')
w=1;Bz=C;time=0;mint=Sumt+1;SumOt=0;MT=[];MN=[];Oz=0;O=[];SEQ=[];
NSEQ=[]; SN=[];SON=[];MB=M
sr=0;GT=zeros(1,1);yg=0;CJ=[];a=0
while tn>sr
 for j=1:tn
   for i=1:pn
     if A(j,i)==-1
        y=M(1,i)
        MT=[MT;y]
     end
   end
   if all(MT==1)
     g=j
     [s1,s2]=size(GT)
     for s=1:s1
        if g==GT(s)
          yg=1
        end
     end
     if yg~=1
       yt(g)=time+t(g)
       Q=[g yt(g)]
       O=[O;Q]
     end
   end
   MT=[];yg=0
 end
 [vr,vc]=size(O)
 for v=1:vr
   if O(v,2)<mint
     mint=O(v,2);Oz=O(v,1);r=v
   end
 end
```

```
 time=mint;MN=A(Oz,:);M=M+MN;mint=Sumt;
 SEQ=[SEQ;Oz]
 [sr,sc]=size(SEQ)
 UO=O(1:r-1,:);AO=O(r+1:vr,:);O=[UO;AO]
 if O==[]
   GT=zeros(1,1)
 else
   GT=O(:,1)
 end
end
NSEQ=SEQ;M=MB;MT=[]
while Sumt>SumOt
 if Bz==0
   Bz=C;w=w+1;SN=[]
 end
 if Bz<t(NSEQ(1,:))
   [nsr,nsc]=size(NSEQ)
   for n=1:nsr
     for i=1:pn
       if A(NSEQ(n),i)==-1
         y=M(1,i)
         MT=[MT;y]
       end
     end
     if all(MT==1)
       CN=[n NSEQ(n)]
       CJ=[CJ;CN]
     end
     MT=[]
   end
   [cr,c2]=size(CJ)
   for cc=1:cr
      if Bz>=t(CJ(cc,2)) & a~=1
        Bz=Bz-t(CJ(cc,2));SumOt=SumOt+t(CJ(cc,2))
        SN=[w CJ(cc,2) t(CJ(cc,2)) Bz];M=M+A(CJ(cc,2),:);a=1
        ANSEQ=NSEQ(1:CJ(cc,1)-1,:)
        UNSEQ=NSEQ(CJ(cc,1)+1:nsr,:);NSEQ=[ANSEQ;UNSEQ]
      end
   end
   if CJ==[] | a==0
     Bz=C;w=w+1;SN=[]
   end
 end
 if Bz>=t(NSEQ(1,:)) & a~=1
   Bz=Bz-t(NSEQ(1,:));SumOt=SumOt+t(NSEQ(1,:))
   M=M+A(NSEQ(1),:);SN=[w NSEQ(1,:) t(NSEQ(1,:)) Bz]
   [nsr,nsc]=size(NSEQ);NSEQ=NSEQ(2:nsr,:)
 end
SON=[SON;SN]
```

```
CJ=[];a=0
end
disp('   St.  Job  Time  Idle')
disp('   ---   ---  ----  ----')
disp(SON)
pause
LE=Sumt/(w*C)
disp('Line efficency of assembl y is(LE)');disp(LE)
```

Appendix 4. MATLAB Code for Firing Sequencing of Transitions-Best Search Procedure

```
clear
disp('This program solves SALBP-1using FS-f')
% A incidence matrix
A=input('input the incidence matrix in the brackets')
[tn,pn]=size(A)
% M initial token conditions
M=input('input initial token conditions in vector format in the brackets')
% t vector which includes times of jobs
t=input('input times of jobs in vector format in the brackets')
t=t'
Sumt=sum(t)
% C cycle time
C=input('input cycle time')
w=1;Bz=C;time=0;mint=Sumt+1;O=[];SumOt=0;MT=[];MN=[];Oz=0;O=[];SEQ=[];
NSEQ=[];SN=[];SON=[];MB=M;sr=0;GT=zeros(1,1);yg=0;CJ=[];a=0;minBz=Bz;
Sw=0;JN=0;JS=0
while tn>sr
 for j=1:tn
   for i=1:pn
     if A(j,i)==-1
       y=M(1,i)
       MT=[MT;y]
     end
   end
   if all(MT==1)
     g=j
     [s1,s2]=size(GT)
     for s=1:s1
       if g==GT(s)
         yg=1
       end
     end
     if yg~=1
       yt(g)=time+t(g)
       Q=[g yt(g)]
       O=[O;Q]
     end
   end
   MT=[];yg=0
 end
 [vr,v2]=size(O)
 for v=1:vr
   if O(v,2)<mint
     mint=O(v,2);Oz=O(v,1);r=v
   end
 end
```

```
 time=mint;MN=A(Oz,:);M=M+MN;mint=Sumt
 SEQ=[SEQ;Oz]
 [sr,sc]=size(SEQ)
 UO=O(1:r-1,:);AO=O(r+1:vr,:);O=[UO;AO]
 if O==[]
   GT=zeros(1,1)
 else
   GT=O(:,1)
 end
end
NSEQ=SEQ;M=MB;MT=[]
while Sumt>SumOt
 if Bz==0
   Bz=C;w=w+1;SN=[];minBz=Bz
 end
 if Bz<t(NSEQ(1,:))
   [nsr,nsc]=size(NSEQ)
   for n=1:nsr
     for i=1:pn
       if A(NSEQ(n),i)==-1
         y=M(1,i)
         MT=[MT;y]
       end
     end
     if all(MT==1)
       CN=[n NSEQ(n)]
       CJ=[CJ;CN]
     end
     MT=[]
   end
   [cr,c2]=size(CJ)
   for cc=1:cr
     if Bz>=t(CJ(cc,2))
       if minBz>=C-Sw-t(CJ(cc,2))
         minBz=C-Sw-t(CJ(cc,2));JN=CJ(cc,2);JS=CJ(cc,1);a=1
       end
     end
   end
   if CJ==[] | a==0
     Bz=C;w=w+1;SN=[];minBz=Bz
   else
     Bz=Bz-t(JN);SumOt=SumOt+t(JN);minBz=Bz
     SN=[w JN t(JN) Bz];M=M+A(JN,:)
      ANSEQ=NSEQ(1:JS-1,:);
      UNSEQ=NSEQ(JS+1:nsr,:);NSEQ=[ANSEQ;UNSEQ]
   end
 end
 if Bz>=t(NSEQ(1,:)) & a==0
   Bz=Bz-t(NSEQ(1,:));SumOt=SumOt+t(NSEQ(1,:));minBz=Bz
```

```
    [nsr,nsc]=size(NSEQ);M=M+A(NSEQ(1):NSEQ(1),:)
    SN=[w NSEQ(1,:) t(NSEQ(1,:)) Bz];NSEQ=NSEQ(2:nsr,:)
 end
Sw=C-Bz
SON=[SON;SN]
CJ=[];a=0
end
disp('   St.  Job  Time  Idle')
disp('   ---  ---  ----  ----')
disp(SON)
pause
LE=Sumt/(w*C)
disp('Line efficency of assembly is(LE)');disp(LE)
```

Appendix 5. MATLAB Code for P-Invariants Procedure

```
clear
disp('This program solves SALBP-1using FS-f')
% A incidence matrix
A=input('input the incidence matrix in the brackets')
[tn,pn]=size(A)
% M initial token conditions
M=input('input initial token conditions in vector format in the brackets')
% t vector which includes times of jobs
t=input('input times of jobs in vector format in the brackets')
t=t'
Sumt=sum(t)
%P P-invariant matrix
P=input('input P-invariants in matrix format in the brackets')
P=P'
[pr,pc]=size(P)
% C cycle time
C=input('input cycle time')
w=1;Sw=0;MT=[];h=[];d=[];F=[];ALS=[];ASW=[];awr=0;SON=[];MB=M
for i=1:pr
   for j=1:pc
      if P(i,j)==1
        h=[h;j]
      end
   end
   [hr,hc]=size(h)
   for k=1:hr-1
      for l=1:tn
         if A(l,h(k))==-1 & A(l,h(k+1))==1
           d=[d;l]
         end
      end
    end
    [dr,dc]=size(d)
    dz=zeros((tn-dr),1);d=[d;dz];d=d'
    F=[F;d];h=[];d=[]
end
F=F'
[fr,fc]=size(F)
while tn>awr
   for ii=1:fc
      M=MB;Swnow=Sw;ww=w
      for jj=1:fr
         a=0
         for zz=1:awr
            if F(jj,ii)==ASW(awr,1)
              a=1
            end
```

```
          end
          if F(jj,ii)>0 & a<1
            for kk=1:pn
              if A(F(jj,ii),kk)==-1
                y=M(1,kk); MT=[MT;y]
              end
            end
            if all(MT==1)
              if C-Swnow>=t(F(jj,ii))
                Swnow=Swnow+t(F(jj,ii))
                AL=[ii ww F(jj,ii) t(F(jj,ii)) Swnow C-Swnow]
                ALS=[ALS;AL]
                B=A(F(jj,ii),:);M=M+B
              end
            end
          end
          MT=[]
        end
      end
      if ALS==[]
        w=w+1;Sw=0
      else
        IST=ALS(:,6)
        [x,z]=min(IST)
        Sw=C-x
        al=ALS(z,1)
        [alr,alc]=size(ALS)
        for aa=1:alr
          if ALS(aa,1)==al
            ASW=[ASW;ALS(aa,3)]
            SON=[SON;w ALS(aa,3) t(ALS(aa,3)) ALS(aa,6)]
            MB=MB+A(ALS(aa,3),:)
          end
        end
        [awr,awc]=size(ASW)
        ALS=[]
      end
    end
    disp('   St.  Job  Time  Idle')
    disp('   ---  ---  ----  ----')
    disp(SON)
    pause
    LE=Sumt/(w*C)
    disp('Line efficency of assembly is(LE)');disp(LE)
```

Appendix 6. QS' Solutions for Talbot's Data Set

Author	Cycle time	# of tasks	Opt.	BBS	FF	FIF	FBA	LBA	LPT	MF	MIF	Rand.	RPWM	SPT	COM.
Bowman	20	8	5	5	5	5	5	5	5	5	5	5	5	5	5
Mansoor	48	11	4	4	5	5	5	4	4	5	5	5	5	5	4
Mansoor	62	11	3	3	4	4	4	4	4	4	3	4	4	4	3
Mansoor	94	11	2	2	3	3	3	2	2	2	3	2	3	3	2
Mertens	6	7	6	6	6	6	6	6	6	6	6	6	6	6	6
Mertens	7	7	5	5	6	6	5	6	5	5	5	6	6	6	5
Mertens	8	7	5	5	6	6	5	6	5	5	5	6	6	6	5
Mertens	10	7	3	4	4	4	4	3	4	3	3	4	4	4	3
Mertens	15	7	2	2	2	2	3	2	2	2	2	2	2	2	2
Mertens	18	7	2	2	2	2	2	2	2	2	2	2	2	2	2
Jaeschke	6	9	8	8	8	8	8	8	8	8	8	8	8	8	8
Jaeschke	7	9	7	7	7	7	7	7	7	7	7	7	7	7	7
Jaeschke	8	9	6	6	6	6	6	6	6	6	6	6	6	6	6
Jaeschke	10	9	4	4	4	4	4	4	4	5	4	5	4	5	4
Jaeschke	18	9	3	3	3	3	3	3	3	3	3	3	3	3	3
Jackson	7	11	8	8	8	8	8	8	8	9	8	9	9	9	8
Jackson	9	11	6	6	6	6	6	6	6	6	6	6	6	7	6
Jackson	10	11	5	6	5	5	6	5	6	6	5	6	6	6	5
Jackson	13	11	4	4	4	4	4	4	4	5	4	4	4	4	4
Jackson	14	11	4	4	4	4	4	4	4	4	4	4	4	4	4
Jackson	21	11	3	3	3	3	3	3	3	3	3	3	3	3	3
Mitchell	14	21	8	9	10	10	10	9	8	8	8	10	8	10	9
Mitchell	15	21	8	9	10	9	9	9	8	8	8	9	8	10	8
Mitchell	21	21	5	5	6	6	6	6	5	6	6	6	6	6	5
Mitchell	26	21	5	5	5	5	5	5	5	5	5	5	5	5	5
Mitchell	35	21	3	3	4	4	4	4	4	4	4	4	4	4	3
Mitchell	39	21	3	3	3	3	3	3	3	3	3	3	3	3	3
Heskiaoff	138	28	8	8	9	9	8	9	8	9	9	9	8	10	8
Heskiaoff	205	28	5	6	7	6	6	6	6	6	6	6	5	6	6
Heskiaoff	216	28	5	5	6	6	6	5	5	5	5	6	5	6	5
Heskiaoff	256	28	4	5	5	5	5	5	5	5	5	5	4	5	4
Heskiaoff	324	28	4	4	4	4	4	4	4	4	4	4	4	4	4
Heskiaoff	342	28	3	3	4	4	4	4	4	3	4	4	3	4	3
Sawyer	25	30	14	14	16	16	15	15	14	15	16	15	15	17	14
Sawyer	27	30	13	13	14	14	14	15	13	14	14	14	14	15	13
Sawyer	30	30	12	12	13	13	12	13	13	13	13	12	12	14	12
Sawyer	36	30	10	10	11	11	10	10	10	10	10	11	10	12	10
Sawyer	41	30	8	9	9	10	10	9	9	9	9	9	9	9	9
Sawyer	54	30	7	7	7	7	7	7	7	7	7	7	7	7	7
Sawyer	75	30	5	5	5	5	5	5	5	5	5	5	5	5	5
Kilbridge	57	45	10	11	11	11	10	11	11	10	11	11	10	11	10
Kilbridge	79	45	7	8	8	8	8	8	8	8	8	8	8	8	8
Kilbridge	92	45	6	7	7	7	7	7	7	7	7	7	7	7	7
Kilbridge	110	45	6	6	6	6	6	6	6	6	6	6	6	6	6
Kilbridge	138	45	4	5	5	5	5	5	5	5	5	5	5	5	4
Kilbridge	184	45	3	3	4	4	4	4	3	4	4	4	4	4	3

Author	Cycle time	# of tasks	Opt.	BBS	FF	FIF	FBA	LBA	LPT	MF	MIF	Rand.	RPWM	SPT	COM.
Tonge	176	70	21	21	25	24	23	24	21	22	24	23	21	26	22
Tonge	364	70	10	10	11	11	10	11	10	10	10	10	10	11	10
Tonge	410	70	9	9	9	9	9	9	9	9	10	9	9	10	9
Tonge	468	70	8	8	8	8	8	8	8	8	8	8	8	8	8
Tonge	527	70	7	7	7	7	8	7	7	7	7	7	7	8	7
Arcus1	5048	83	16	16	17	17	16	17	16	16	17	16	16	17	16
Arcus1	5853	83	14	14	14	14	14	15	14	14	14	14	14	14	14
Arcus1	6842	83	12	12	13	13	13	13	13	12	13	12	12	13	12
Arcus1	7571	83	11	11	11	11	11	11	11	11	11	11	11	11	11
Arcus1	8412	83	10	10	10	10	10	10	10	10	10	10	10	10	10
Arcus1	8898	83	9	9	9	9	9	9	9	9	9	9	9	9	9
Arcus1	10816	83	8	8	8	8	8	8	8	8	8	8	8	8	8
Arcus2	5755	111	27	27	30	29	29	30	27	28	29	29	27	34	28
Arcus2	8847	111	18	18	21	19	19	19	18	18	18	19	18	21	18
Arcus2	10027	111	16	16	17	17	17	16	16	16	16	17	16	17	16
Arcus2	10743	111	15	15	16	16	16	16	15	15	15	16	15	17	15
Arcus2	11378	111	14	14	15	15	15	15	14	14	14	15	14	15	14
Arcus2	17067	111	9	9	10	10	9	10	9	9	9	9	9	10	9

Appendix 7. QS' Solutions for Hoffmann's Data Set

Author	Cycle time	# of tasks	Opt.	BBS	FF	FIF	FBA	LBA	LPT	MF	MIF	Rand.	RPWM	SPT	COM.
Sawyer	27	30	13	13	14	14	14	15	13	14	14	14	14	15	13
Sawyer	30	30	12	12	13	13	12	13	13	13	13	12	12	14	12
Sawyer	33	30	11	11	12	11	12	11	11	11	12	12	11	13	11
Sawyer	36	30	10	10	11	11	10	10	10	10	10	11	10	12	10
Sawyer	41	30	8	9	9	10	10	9	9	9	9	9	9	9	9
Sawyer	47	30	7	8	8	8	8	8	8	8	8	8	8	9	8
Sawyer	54	30	7	7	7	7	7	7	7	7	7	7	7	7	7
Kilbridge	56	45	10	11	11	11	11	11	11	10	11	11	10	11	10
Kilbridge	62	45	9	10	10	10	10	10	10	9	10	10	10	11	10
Kilbridge	69	45	8	9	10	10	9	9	9	9	9	9	9	10	9
Kilbridge	79	45	7	8	8	8	8	8	8	8	8	8	8	8	8
Kilbridge	92	45	6	7	7	7	7	7	7	7	7	7	7	7	7
Kilbridge	111	45	5	5	6	6	5	6	6	5	5	6	5	6	5
Tonge	160	70	23	23	26	24	24	26	23	24	25	26	24	27	24
Tonge	168	70	22	23	26	24	23	25	23	23	24	24	23	27	23
Tonge	176	70	21	21	25	24	23	24	21	22	24	23	21	26	22
Tonge	185	70	20	21	23	22	22	22	21	20	21	22	20	23	20
Tonge	195	70	19	20	22	21	20	22	20	20	20	21	19	22	19
Tonge	207	70	18	18	20	19	20	20	18	18	19	20	18	21	18
Tonge	220	70	17	18	19	19	18	18	18	17	18	18	17	20	17
Tonge	234	70	16	16	17	17	17	17	16	16	16	17	16	20	16
Tonge	251	70	14	15	16	16	16	16	15	15	15	16	15	17	15
Tonge	270	70	14	14	14	14	15	14	14	14	14	15	14	16	14
Tonge	293	70	13	13	13	13	13	14	13	13	13	13	13	14	13
Tonge	320	70	11	12	13	12	12	12	12	12	12	12	12	13	12
Arcus1	3786	83	21	21	24	23	22	23	22	22	23	22	22	24	22
Arcus1	3985	83	20	20	23	21	21	22	21	21	23	21	21	23	21
Arcus1	4206	83	19	20	22	20	20	21	20	20	21	20	19	21	19
Arcus1	4454	83	18	18	19	19	20	19	18	19	21	20	19	20	18
Arcus1	4732	83	17	17	18	18	18	18	17	18	18	18	18	18	17
Arcus1	5048	83	16	16	17	17	16	17	16	16	17	16	16	17	16
Arcus1	5408	83	15	15	15	15	15	15	16	15	15	16	15	15	15
Arcus1	5824	83	14	14	14	14	14	14	14	14	14	14	14	14	14
Arcus1	6309	83	13	13	13	14	13	14	13	13	13	13	13	14	13
Arcus1	6883	83	12	12	13	13	13	13	12	12	13	13	12	13	12
Arcus1	7571	83	11	11	11	11	11	11	11	11	11	11	11	11	11
Arcus2	5785	111	27	27	30	29	29	30	27	28	29	29	27	34	28
Arcus2	6016	111	26	26	30	29	28	28	26	27	28	27	27	34	27
Arcus2	6267	111	25	25	28	28	27	27	25	25	27	27	26	30	26
Arcus2	6540	111	24	25	27	27	27	26	25	25	25	27	24	28	25
Arcus2	6837	111	23	24	26	27	25	25	24	23	24	25	23	26	24
Arcus2	7162	111	22	22	25	25	24	24	22	22	23	24	22	25	23
Arcus2	7520	111	21	21	23	24	23	23	22	21	23	23	21	25	22
Arcus2	7916	111	20	20	23	22	22	22	20	20	21	22	20	25	21
Arcus2	8356	111	19	19	21	20	20	20	19	19	20	21	19	22	19
Arcus2	8847	111	18	18	21	19	19	19	18	18	18	19	18	21	18
Arcus2	9400	111	17	17	19	18	18	17	17	17	18	18	17	20	17
Arcus2	10027	111	16	16	17	17	17	16	16	16	16	17	16	17	16
Arcus2	10743	111	15	15	16	16	16	16	15	15	15	16	15	17	15
Arcus2	11570	111	13	14	15	15	14	14	14	14	14	14	14	15	14

Appendix 8. Solutions Obtained by LBHA-1 and the Proposed Algorithms

Author	Cycle time	# of tasks	Opt. (B)*	Opt. (S&K)*	LBHA-1	RA	FS	FS-f	FS-b	P-in
Mertens	9	7	4	4	4	4	4	4	4	4
Mertens	10	7	3	3	3	4	4	4	4	4
Mertens	15	7	2	2	2	2	2	2	2	2
Bowman	20	8	5	5	5	5	5	5	5	5
Jaeschke	6	9	8	8	8	8	8	8	8	8
Jaeschke	7	9	7	7	7	7	7	7	7	7
Jaeschke	8	9	6	6	6	6	7	6	6	6
Jaeschke	9	9	5	5	5	5	5	5	5	5
Jaeschke	10	9	4	4	4	4	4	4	4	5
Jaeschke	18	9	3	3	3	3	3	3	3	3
Jackson	8	11	7	7	7	7	8	8	8	7
Jackson	9	11	6	6	6	6	6	6	6	6
Jackson	10	11	5	5	5	6	6	6	6	6
Jackson	12	11	4	4	4	5	5	5	5	5
Jackson	17	11	3	3	3	3	3	3	3	3
Jackson	24	11	2	2	2	2	2	2	2	2
Mitchell	18	21	6	6	6	6	7	7	7	7
Mitchell	19	21	6	6	6	6	7	7	7	6
Mitchell	23	21	5	5	5	5	5	5	5	5
Mitchell	35	21	4	3	4	3	4	4	4	3
Mitchell	39	21	3	3	3	3	3	3	3	3
Heskiaoff	205	28	5	5	5	6	6	6	6	6
Heskiaoff	216	28	5	5	5	5	6	6	6	5
Heskiaoff	324	28	4	4	4	4	4	4	4	4
Heskiaoff	342	28	3	3	3	4	4	4	4	3
Sawyer	25	30	15	14	15	14	17	15	15	15
Sawyer	27	30	13	13	14	13	16	15	14	14
Sawyer	28	30	13	13	13	13	16	15	14	13
Sawyer	32	30	12	11	12	12	13	13	13	11
Sawyer	33	30	11	11	11	11	13	13	13	11
Sawyer	36	30	10	10	10	10	11	11	11	10
Sawyer	39	30	9	9	9	9	10	10	10	9
Sawyer	41	30	8	8	9	9	10	10	10	9
Sawyer	42	30	8	8	8	9	10	10	10	9
Sawyer	54	30	7	7	7	7	7	7	7	7
Sawyer	75	30	5	5	5	5	5	5	5	5
Kilbridge	57	45	10	10	10	10	12	11	11	10
Kilbridge	79	45	8	7	8	8	8	8	8	8
Kilbridge	92	45	7	6	7	7	7	7	7	7
Kilbridge	110	45	6	6	6	6	6	6	6	6
Kilbridge	138	45	5	4	5	5	5	5	5	5
Kilbridge	184	45	4	3	5	3	4	4	4	4
Arcus1	5048	83	16	16	16	16	17	17	17	16
Arcus1	5853	83	14	14	14	14	14	14	14	14
Arcus1	6842	83	12	12	12	12	13	13	13	12
Arcus1	7571	83	11	11	11	11	11	11	11	11
Arcus1	8412	83	10	10	10	10	10	10	10	10
Arcus1	8898	83	9	9	9	9	9	9	9	9
Arcus1	10816	83	8	8	8	8	8	8	8	8

Note: Opt. (B)* and Opt. (S&K)* are the optimum solutions according to the Baybars and Scholl & Klein, respectively.

Appendix 9. Solutions Obtained by Chang's Algorithm and the Proposed Algorithms

Author	Cycle time	# of tasks	Opt.	CHANG'S				PROPOSED ALG.				
				RBI with TA	RBI without TA	RFI with TA	RFI without TA	RA	FS	FS-f	FS-b	P-in
Mertens	6	7	6	6	6	6	6	6	6	6	6	6
Mertens	7	7	5	5	5	5	5	5	6	6	6	5
Mertens	8	7	5	5	5	5	5	5	6	6	6	5
Mertens	10	7	3	3	3	3	3	4	4	4	4	4
Mertens	15	7	2	2	2	2	2	2	2	2	2	2
Mertens	18	7	2	2	2	2	2	2	2	2	2	2
Bowman	20	8	5	5	5	5	5	5	5	5	5	5
Jaeschke	6	9	8	8	8	8	8	8	8	8	8	8
Jaeschke	7	9	7	7	7	7	7	7	7	7	7	7
Jaeschke	8	9	6	7	6	7	6	6	7	6	6	6
Jaeschke	10	9	4	4	4	4	4	4	4	4	4	5
Jaeschke	18	9	3	3	3	3	3	3	3	3	3	3
Jackson	7	11	8	8	8	8	8	8	9	9	9	8
Jackson	9	11	6	6	6	6	6	6	6	6	6	6
Jackson	10	11	5	5	5	5	5	6	6	6	6	6
Jackson	13	11	4	4	4	4	4	4	5	5	5	4
Jackson	14	11	4	4	4	4	4	4	4	4	4	4
Jackson	21	11	3	3	3	3	3	3	3	3	3	3
Mitchell	14	21	8	8	8	9	8	9	10	10	10	9
Mitchell	15	21	8	8	8	8	8	8	10	10	10	9
Mitchell	21	21	5	5	5	5	5	5	6	6	6	6
Mitchell	26	21	5	5	5	5	5	5	5	5	5	5
Mitchell	35	21	3	3	3	3	3	3	4	4	4	3
Mitchell	39	21	3	3	3	3	3	3	3	3	3	3
Heskiaoff	138	28	8	9	8	9	8	8	9	8	8	8
Heskiaoff	205	28	5	5	5	6	5	6	6	6	6	6
Heskiaoff	216	28	5	5	5	5	5	5	6	6	6	5
Heskiaoff	256	28	4	4	4	4	4	5	5	5	5	5
Heskiaoff	324	28	4	4	4	4	4	4	4	4	4	4
Heskiaoff	342	28	3	3	3	3	3	4	4	4	4	3
Sawyer	25	30	14	14	14	14	14	14	17	15	15	15
Sawyer	27	30	13	13	13	13	13	13	16	15	14	14
Sawyer	30	30	12	12	12	12	12	13	13	13	13	12
Sawyer	36	30	10	10	10	10	10	10	11	11	11	10
Sawyer	41	30	8	8	8	8	8	9	10	10	10	9
Sawyer	54	30	7	7	7	7	7	7	7	7	7	7
Sawyer	75	30	5	5	5	5	5	5	5	5	5	5
Kilbridge	57	45	10	10	10	10	10	10	12	11	11	10
Kilbridge	79	45	7	8	7	7	7	8	8	8	8	8
Kilbridge	92	45	6	7	6	6	6	7	7	7	7	7
Kilbridge	110	45	6	6	6	6	6	6	6	6	6	6
Kilbridge	138	45	4	5	4	5	4	5	5	5	5	5
Kilbridge	184	45	3	3	3	3	3	3	4	4	4	4

Author	Cycle time	# of tasks	Opt.	CHANG'S RBI with TA	RBI without TA	RFI with TA	RFI without TA	PROPOSED ALG. RA	FS	FS-f	FS-b	P-in
Tonge	176	70	21	23	21	23	21	22	26	24	24	23
Tonge	364	70	10	11	11	11	11	10	12	11	11	10
Tonge	410	70	9	9	9	9	9	9	10	10	10	9
Tonge	468	70	8	9	8	8	8	8	8	8	8	8
Tonge	527	70	7	7	7	7	7	7	8	8	8	7
Arcus1	5048	83	16	18	17	18	16	16	17	17	17	16
Arcus1	5853	83	14	15	14	15	14	14	14	14	14	14
Arcus1	6842	83	12	13	12	13	12	12	13	13	13	12
Arcus1	7571	83	11	11	11	11	11	11	11	11	11	11
Arcus1	8412	83	10	10	10	10	10	10	10	10	10	10
Arcus1	8898	83	9	10	9	10	9	9	9	9	9	9
Arcus1	10816	83	8	8	8	8	8	8	8	8	8	8
Arcus2	5755	111	27	28	27	28	27	27	33	31	31	27
Arcus2	8847	111	18	18	19	19	18	18	20	19	19	18
Arcus2	10027	111	16	16	16	16	16	16	18	17	17	16
Arcus2	10743	111	15	15	15	15	15	15	17	17	17	15
Arcus2	11378	111	14	14	14	14	15	14	15	15	15	14
Arcus2	17067	111	9	10	9	10	10	9	10	10	10	9